

T Nelson Jan 73

COMPUTER-AUGMENTED MANAGEMENT-SYSTEM RESEARCH AND DEVELOPMENT OF AUGMENTATION FACILITY

D. C. ENGELBART and
STAFF OF AUGMENTATION RESEARCH CENTER

Stanford Research Institute

Distribution of this document is unlimited. It may be released to the Clearing-house, Department of Commerce, for sale to the general public.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by D. Stone, RADC (EMBIH), GAFB, NY 13440 under Contract No. F30602-68-C-0286.

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded, by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

RADC-TR-70-82
Final Report
April 1970

COMPUTER-AUGMENTED MANAGEMENT-SYSTEM RESEARCH AND DEVELOPMENT OF AUGMENTATION FACILITY

Contractor: Stanford Research Institute
Contract Number: F30602-68-C-0286
Effective Date of Contract: 10 April 1968
Contract Expiration Date: 10 April 1970
Amount of Contract: \$1,515,222
Program Code Number: 8D30

Principal Investigator: Dr. D. C. Engelbart
Phone: 415 326-6200 Ext 2220

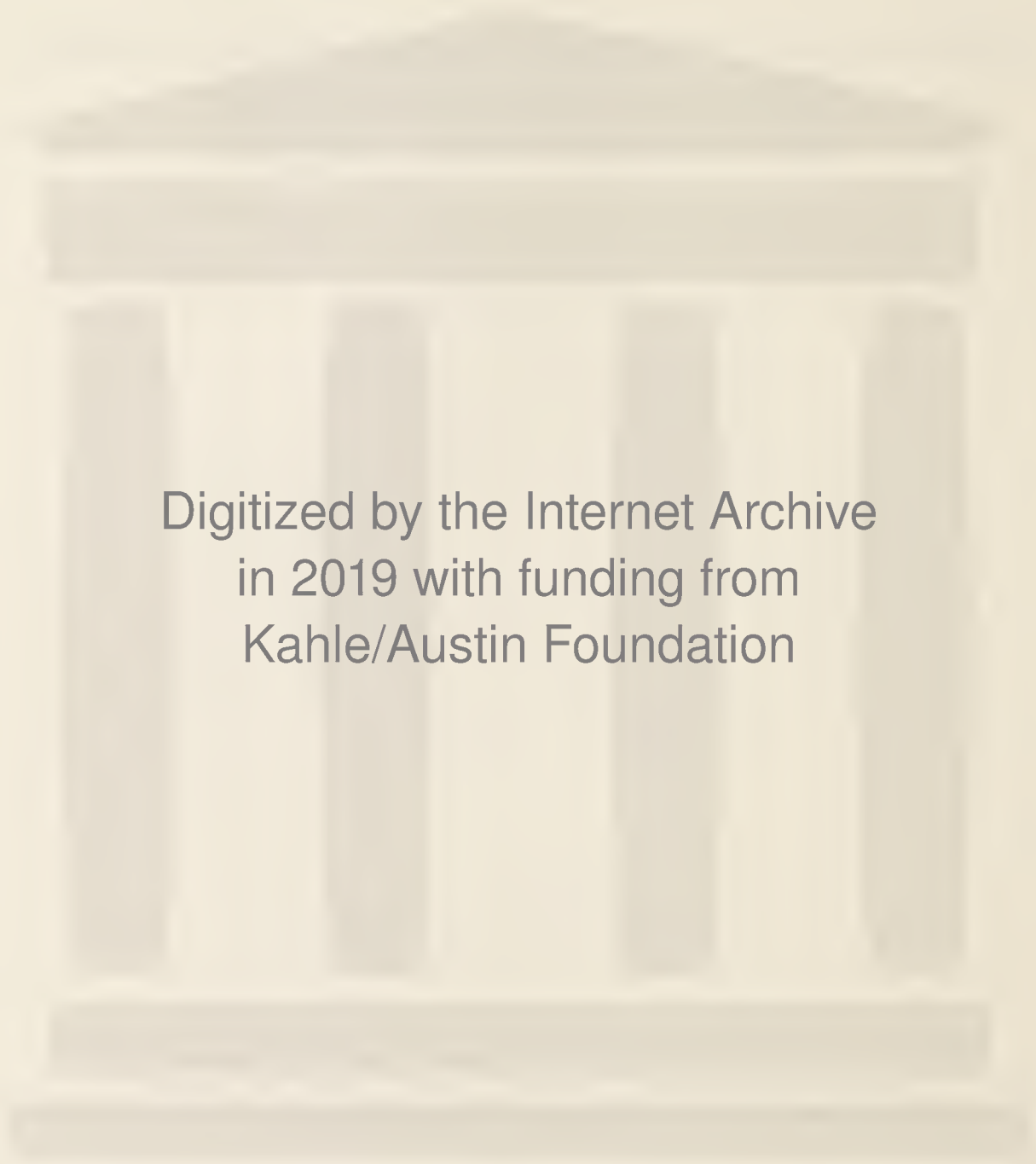
Project Engineer: D. Stone
Phone: 315 330-2600

Sponsored by
ADVANCED RESEARCH PROJECTS AGENCY
ARPA Order No. 0967

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK



Digitized by the Internet Archive
in 2019 with funding from
Kahle/Austin Foundation

<https://archive.org/details/computeraugmente00dcen>

ABSTRACT

This report covers two years of research in a continuing program in the Augmentation Research Center (ARC) of the Information Sciences Laboratory of Stanford Research Institute, supported by ARPA and RADG under Contract F30602-68-C-0286.

Some of the work reported was also supported by ARPA and NASA under Contract NAS1-7897.

The research reported is aimed at the development of on-line computer aids for increasing the performance of individuals and teams engaged in intellectual work, and the development of techniques for the use of such aids. The report covers hardware and software development, applications in several areas relating to management of a community of workers who use on-line aids and to information management for such a community, participation in the ARPA computer network, and a summary of plans for the continuation of the research.

PREFACE

The research described in this report represents conceptual, design, and development work by a large number of people; the program has been active as a coordinated team effort since 1963. The research reported here was a cooperative team effort involving the entire ARC staff. The following is an alphabetical listing of the current ARC staff:

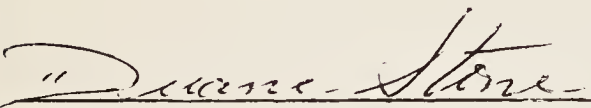
Geoffrey H. Ball, Walter L. Bass, Vernon R. Baughman, Mary G. Caldwell, Roberta A. Carillon, David Casseres, Mary S. Church, William S. Duvall, Douglas C. Engelbart, William K. English, Ann R. Geoffrion, Martin L. Hardy, Jared M. Harris, J. David Hopper, Charles H. Irby, L. Stephen Leonard, John T. Melvin, N. Dean Meyer, James C. Norton, Bruce L. Parsley, William H. Paxton, Jake Ratliff, Barbara E. Row, Martha E. Trundy, Edward K. Van de Riet, John M. Yarborough.

The following former ARC staff members also contributed to the research:

Donald I. Andrews, Roger D. Bates, David A. Evans, Stephen R. Levine, Stephen H. Paavola, Helen M. Prince, Jons F. Rulifson, Elmer B. Shapiro, F. K. Tomlin.

PUBLICATION REVIEW

This technical report has been reviewed and is approved.


RADC Project Engineer

TECHNICAL EVALUATION

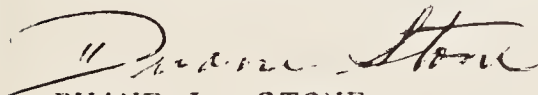
The Augmentation Research Center (ARC) is a community of about 28 researchers, supported by several different contracts since 1963, in which all the research activity is aimed at (1) exploring the possibilities for augmenting the performance of intellectual work with the help of real-time computer aids and (2) the experimental development of computer aids and augmentation systems.

All the researchers within the ARC do as much of their work as possible at display consoles (depending on console availability and whether a specific task can appropriately be done at a console). Thus they serve not only as researchers but as the subjects for the analysis and evaluation of the augmentation systems that they are developing.

Consequently, an important aspect of the augmentation work done within the ARC is that the techniques being explored are implemented, studied, and evaluated with the advantage of intensive everyday usage within a coordinated working environment that is compatible with the particular techniques being studied. This strategy, called "bootstrapping," is a key concept in much of the ARC design philosophy.

The focus of the augmentation is on "text" manipulation, where text is defined as strings of characters, mathematical equations, programming statements, line drawings, columns of figures, etc. A powerful set of commands allow instantaneous composition, editing, copying, printing, analysis, calculation, etc. through interaction via a TV display, binary keyset, keyboard, and display pointing device.

The system is successfully used at the ARC in all phases of daily activity including: program writing and debugging, report preparation and printing, conducting meetings and demonstration, project management, note taking, etc. At least part of the success of the system is due to the dedication and zeal with which the ARC personnel use and develop it.



DUANE L. STONE
Technical Evaluator

CONTENTS

ABSTRACT.....	1
PREFACE.....	iii
TECHNICAL EVALUATION BY SPONSOR.....	v
LIST OF ILLUSTRATIONS.....	ix
I INTRODUCTION.....	1
II MANAGEMENT SYSTEM.....	5
A. Management-Information Operations.....	5
1. Introduction.....	5
2. Project Costs.....	5
3. Activity Planning and Status.....	32
B. Organization Studies.....	36
1. On-Line Community.....	37
2. Experiments on Internal Activity Structure.....	41
3. Observations From Study of On-Line Community.....	47
C. Team Augmentation and Dialogue Support.....	50
1. Recent Efforts.....	50
2. Future Approaches to Team Augmentation.....	50
III HARDWARE SYSTEM.....	57
A. Introduction.....	57
B. The Computer Facility.....	57
C. Modifications in Progress.....	62
D. Notes on System Design and Reliability.....	66
IV SOFTWARE SYSTEM.....	77
A. Introduction.....	77
B. Timesharing System.....	80
C. Compilers.....	82
D. Response Studies.....	94
E. The On-line System, NLS.....	109
F. ARPA Computer Network.....	119
F. NLS Utility Subsystem.....	123
V FUTURE PLANS.....	127
A. General.....	127
B. Shifts in Emphasis.....	127
C. Transfer of Results.....	129
D. Short-Term and Long-Term Goals.....	131
E. Selected Plans Under Other Sponsorship.....	131
GLOSSARY.....	133
REFERENCES.....	135
BIBLIOGRAPHY.....	137
Appendix A: USER FEATURES OF NLS AND TODAS.....	139
Appendix B: DIALOGUE SUPPORT SYSTEM (DSS).....	157
Appendix C: REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT.....	183
Appendix D: TECHNICAL DESCRIPTION OF NLS/TODAS IMPLEMENTATION.....	199
DD Form 1473.....	269

ILLUSTRATIONS

Fig. II-1	A Branch of File HISCO.....	8
Fig. II-2	A Branch of File HISCO.....	8
Fig. II-3	A Branch of File HISCO.....	9
Fig. II-4	A Branch of File HISCO.....	9
Fig. II-5	A Branch of File HISCO.....	10
Fig. II-6	Initial View of File HISCO Upon Entry via Link.....	10
Fig. II-7	A Branch of File COSTS, Showing Entries for 4-Week Accounting Periods.....	13
Fig. II-8	Same as Fig. II-7, but Expanded to Show Weekly Entries.....	13
Fig. II-9	Same as Fig. II-8 but for a Different Branch of File COSTS Showing Data for a Different Project.....	14
Fig. II-10	A Branch of File COSTS Showing Combined Data for all APC Projects.....	14
Fig. II-11	Initial View of File COSTS Upon Entry via Link.....	16
Fig. II-12	Same as Fig. II-11 but with Different VIEWSPECS to Show Content-Analyzer Patterns Stored in First Statement of File.....	16
Fig. II-13	View of File COSTS with Content Analyzer in Operation, Showing Data for Only a Single Week (This is done by using the first pattern appearing in square brackets in Fig. II-12.).....	18
Fig. II-14	Same as Fig II-13, but After a User Has Inserted Cumulative Totals in the Columns.....	18
Fig. II-15	View of a User's File Directory, Showing First-Level Statements Only.....	20
Fig. II-16	Same as Fig. II-15 but with all Levels Displayed.....	20

Fig. II-17	Part of a File Containing Information on ARC Personnel (Not all levels are shown.).....	21
Fig. II-18	A View Obtained by Jumping to One of the Statements Shown in Fig. II-17 and Opening an Additional Level.....	21
Fig. II-19	A View Obtained by Jumping to the Last Statement Shown in Fig. II-18, with no Change in VIEWSPeCs.....	22
Fig. II-20	Content-Analyzer Patterns Stored in the Personnel-Information File (Each set of square brackets contains one pattern, used to search for hidden "tags" in statements in the file.).....	22
Fig. II-21	View Obtained by Using Content Analyzer to Select Entries in Personnel-Information File that Are Tagged for "Hardware".....	24
Fig. II-22	View Obtained by Using Content Analyzer to Select Entries in Personnel-Information File that Are Tagged for "Software".....	24
Fig. II-23	Part of an On-Line Cost Estimate for Use in a Proposal.....	26
Fig. II-24	Part of an On-Line Cost Estimate for Use in a Proposal.....	26
Fig. II-25	Part of an On-Line Cost Estimate for Use in a Proposal.....	27
Fig. II-26	View of a Portion of the Purchase-Order Processing File, Showing Contents of Individual Statements.....	27
Fig. II-27	View of a Portion of the Purchase-Order Processing File, Showing Outstanding Orders Located in a Separate Branch -- Upper Part of Screen Shows a Branch Containing Content-Analyzer Patterns.....	30
Fig. II-28	A Content-Analyzer Pattern for Searching in the Purchase-Order File.....	30
Fig. II-29	View Generated by a Search on the Pattern Shown in Fig. II-28.....	31
Fig. II-30	Task Milestone Chart from File UPLAN.....	35

Fig. II-31	Top-Level View of File UMEET, Showing Accumulation of Notes from a Series of Meetings in a Single File.....	35
Fig. II-32	Views of Consoles in Use in the ARC Work Area.....	39
Fig. III-1	XDS940 Computer Facility.....	58
Fig. III-2	Special Devices Channel.....	59
Fig. III-3	Special Devices Channel with External Core....	64
Fig. III-4	Network Interface Construction, Showing Mounting Systems for Circuit Arrays and Multiplex Switch.....	72
Fig. IV-1	Current System: Average and 80-Percent Delays for NLS Input-Feedback and File-Reference Tasks -- Users Equally Divided Between NLS and TODAS.....	97
Fig. IV-2	Percentage of Time Spent in Various System Functions -- Users Equally Divided Between NLS and TODAS.....	98
Fig. IV-3	System with and without QNL: Distribution of Delay Times (in Seconds) for NLS File-Reference Tasks -- 3 NLS Users, 3 TODAS Users, 1 OTHER User.....	100
Fig. IV-4	System With QNL and New Drums: Average and 80-Percent Times for NLS Input-Feedback and File-Reference Tasks With 1 OTHER User and Remaining Users Evenly Divided between NLS and TODAS.....	102
Fig. IV-5	Current System With Various CPU Speeds Relative to Current System CPU: 80-Percent Times for NLS File-Reference Tasks -- Users Equally Divided Between NLS and TODAS.....	104
Fig. IV-6	Current System With Various Core Sizes: 80-Percent Times for NLS File-Reference Tasks -- Users Equally Divided Between NLS and TODAS.....	105
Fig. IV-7	System With QNL and New Drums, With and Without IDS: 80-Percent Times for NLS File-Reference Tasks -- 1 OTHER User, Remaining Users Equally Divided between NLS and TODAS.....	107

Fig. IV-8 System With QNL and New Drums, with and
 Without IDS: 80-Percent Times for Sequence
 of 3 Input-Feedback Tasks and 1
 File-Reference Task -- 1 OTHER User,
 Remaining Users Equally Divided
 Between NLS and TOLAS.....100

Fig. IV-9 Logical Organization of NLS.....111

I INTRODUCTION

A. General

The Augmentation Research Center (ARC) is a community of about 28 researchers, supported by several different contracts, in which all the research activity is aimed at (1) exploring the possibilities for augmenting the performance of intellectual work with the help of real-time computer aids and (2) the experimental development of computer aids and augmentation systems.

Several different coordinated research activities have been developed, sponsored by different contracts, to pursue the various aspects of this augmentation research. The aspects reported here are:

(1) The Management System Research Activity, which has been supported by RADC under this contract.

(2) The development, operation, and maintenance of a real-time computer-display system, including both hardware and software aspects and participation in the ARPA computer network experiment. This has been supported by ARPA and RADC under this contract, and by ARPA and NASA under Contract NAS1-7897. The facility is dedicated solely to the ARC's activities.

All the researchers within the ARC do as much of their work as possible at display consoles (depending on console availability and whether a specific task can appropriately be done at a console). Thus they serve not only as researchers but as the subjects for the analysis and evaluation of the augmentation systems that they are developing.

Consequently, an important aspect of the augmentation work done within the the ARC (for instance, of the RADC-supported Management Systems Research) is that the techniques being explored are implemented, studied, and evaluated with the advantage of intensive everyday usage within a coordinated working environment that is compatible with the particular techniques being studied.

This strategy, called "bootstrapping," is a key concept in much of our design philosophy.

B. On-Line Aid Systems in the Augmentation Research Center

This section very briefly describes the two major augmentation systems available to workers in the Augmentation Research Center. These systems are the On-Line System (NLS) and the Typewriter-Oriented Documentation-Aid System (TODAS).

Appendix A is a more complete description of the user features of these systems; the reader who is not already acquainted with

Sec. I INTRODUCTION

ARC's research will find that this appendix provides a useful background for the main body of the report.

In addition, Appendix D gives a detailed description of NLS/TODAS implementation.

1. The On-Line System (NLS)

NLS, as currently implemented, is essentially a highly interactive, display-oriented text-manipulation system.

NLS is intended to be used on a regular, more or less full-time basis in a time-sharing environment, by users who are not necessarily computer professionals. The practices and techniques developed by users for exploiting NLS are as much a subject of research interest as the development of NLS itself.

a. Structured Text

All text handled by NLS is in "structured-statement" form. This special format is simply a hierarchical arrangement of "statements," resembling a conventional "outline" form.

A statement is simply a string of text, of any length; this serves as the basic unit in the construction of the hierarchy. Each paragraph and heading in this document is an NLS statement.

b. Use of the System

The creation of new text material as content for a file is achieved by typing the new material on a keyboard, under any of several possible NLS commands.

The study capabilities of NLS constitute its most powerful and unusual features. The following is a brief, condensed description of the operations that are possible.

The process of moving from one point in an NLS file to another, which corresponds to turning pages in hard copy, is called "jumping." A very large family of "jump" commands allows the user to specify locations in the file in a number of ways -- e.g., by specifically identifying a statement or by specifying a structural relationship to some other statement.

The NLS content analyzer permits automatic searching of a file for statements satisfying some content pattern

Sec. I INTRODUCTION

specified by the user. The pattern is written in a special language as part of the file text.

A large repertoire of editing commands is provided for modification of the text in a file.

2. The Typewriter-Oriented Documentation-Aid System (TODAS)

TODAS is a text-handling system designed as a "typewriter" counterpart to NLS. TODAS can be operated from a Teletype or any other kind of hard-copy terminal, including terminals linked to the ARC timesharing computer facility (an XDS 940 with special hardware) through acoustic couplers and ordinary telephone lines (as opposed to NLS, which requires microwave transmission to achieve the necessary bandwidth for displays).

3. Output Facilities

The facilities for producing hard-copy output from NLS/TODAS files include a line printer, a paper-tape-driven typewriter, and the Graphics-Oriented Document Output System (GODOS).

The line printer, because of its speed of operation, is the routine means of producing hard copy for use within ARC. It is used heavily by all NLS/TODAS researchers.

The paper-tape typewriter is used for producing report-quality typing, such as this report. As it is relatively slow and inconvenient, it is not normally used except for final output of material to be published.

GODOS produces magnetic tape which is then turned over to an out-of-house facility where it is run on Stromberg-Carlson microfilm equipment to produce frames of microfilm (or microfiche) corresponding to pages of full-size hard copy. The advantage of this system is that it can handle drawings produced in NLS files by means of the NLS graphics capability. GODOS is still in the experimental stage and has not been used extensively.

4. This Report as an Example of NLS/TODAS Capability

The following discussion may be taken as a very rough indication of the power of NLS and TODAS as applied to a single specific problem -- namely, the writing, editing, and production of this report.

The above descriptions of NLS and TODAS were produced by

Sec. I
INTRODUCTION

modification, using NLS, of the more detailed descriptions in Appendix A.

The entire task of modification, including formatting, insertion into the body of the report, and all other details, required about half an hour of work by an NLS user who was already familiar with the contents of the descriptions. If the job had been done by someone who was not familiar with the material (but who was familiar with NLS) it might have taken fifteen minutes longer.

The original description was written for an earlier report and then kept available as an NLS/TODAS file in anticipation of future opportunities for using it.

Indeed, a considerable amount of the material in this report was developed by modification of existing files, and we may expect the new material generated for this report to continue in use as a collection of NLS/TODAS files for as long as it can be updated to reflect current reality.

TODAS was used primarily for the task of entering new material into on-line files. Considerable portions of the material were put on line by a secretary using TODAS, working from handwritten material and from recorded dictation.

Finally, we may note that the writing of this report, using NLS and TODAS throughout, was achieved under considerable time pressure by a team consisting of about a dozen people, all of whom were doing other important work at the same time.

II MANAGEMENT SYSTEM

Our Management System Research Activity has involved three major areas of concentration. In practice these areas overlap considerably, so that there is an integrated research effort on many phases of management technique and theory that impinge upon the operation of ARC. For purposes of description, however, we discuss each area of concentration as if it were an independent effort.

The three areas are:

- (1) Management-Information Operations -- research on techniques for using management information in the ARC environment, including the development of computer aids for the storage and manipulation of such information
- (2) Organization Studies -- research on the ARC on-line community of workers and experimentation with organization structure and planning methods in the on-line community
- (3) Team Augmentation and Dialogue Support-- research on augmenting a team or community of intellectual workers by means of systems that support the intellectual dialogue of the team.

A. Management-Information Operations

1. Introduction

In accordance with our usual strategy, we have pursued our investigation of management-information operations by using NLS and TODAS to develop and provide aids for management of the ARC on-line community.

There are many areas of potential application for on-line aids; we have chosen those which appear to be most useful operationally for experiments with the development of on-line aids.

This section gives detailed descriptions of several applications that have been developed, illustrated with photographs of the NLS display screens to show sequences of information-manipulation operations. A familiarity with the basics of NLS is assumed; Appendix A is intended to provide the necessary information about NLS.

In following the descriptions, it is worth keeping in mind that the speed with which NLS serves its users is an important part of its utility. The photographs indicate transitions that normally take only one or two seconds. This speed lends great power and flexibility to the relatively simple service functions performed by NLS.

2. Project Costs

The most obvious area for application of on-line aids to management within ARC is project cost accounting. Considerable work has been done on the development of several cost-information files and on techniques for their use.

a. Cost Records

The Institute's accounting system provides ARC with detailed cost records for the various "SRI projects" (i.e., individual contracts) being carried out in ARC.

The primary inputs to SRI's system are (1) weekly time cards reporting hourly charges to various projects by individual staff members, and (2) non-labor costs charged directly to projects, including actual charges to projects and commitments (uncompleted orders).

For each SRI project, the accounting system computes dollar costs based on actual salary data for each staff member's hours charged, adds payroll burden and overhead amounts at current rates, combines these costs with non-labor totals, adds appropriate fees, and totals all such charges each week on a cumulative basis.

Current charges are reported to ARC each week on the Project Status Report.

We need frequent and rapid access to project cost summary data for operational use, with less reference to lower-level details, except as the costs are first checked for reasonableness and accuracy. Therefore we decided to start by putting summary data on-line at ARC. As needed in the future, we can add more levels of detail.

File HISCO

We first constructed a cost-history file for 1968-1969 costs on SRI projects ESU 7101 (RADC Contract F30602-68-C-0286) and ESU 7079 (NASA Contract NAS 1-7897). This file is called HISCO.

We decided that the elements of HISCO would include the following for each of the two projects, on the basis of 4-week accounting periods (as used by SRI's accounting system):

- (a) Salary
- (b) Burden
- (c) Overhead
- (d) Total cost
- (e) Fee
- (f) Total charges.

See Figs. II-1, II-2, and II-3. Each of these figures shows a display of one branch of the file, containing the information for a specific project and year.

We also needed a section showing combined salary costs and combined total charges for all of our projects (see Figs. II-4 and II-5). We put these costs in separate branches of the file. The last branch shows total costs for both projects combined. We retroactively studied existing records for all 1968 data and kept up the 1969 costs every 4 weeks, entering the new data by hand.

We experimented with the use of graphic representations by entering charts in HISCO. These charts showed the cumulative cost trends for each project in a separate branch of the file.

We established links between tabular data and chart projections. This made it quite easy to refer to both formats alternately.

The use of graphics in HISCO gave some indication of the usefulness of such linking, but the existing package has limitations in the form of a few bugs and capacity that makes its use of marginal value. Work is currently under way to improve this capability. We also need local hard-copy output to make these features of real value.

HISCO was a testing ground for the first version of the NLS calculator package. As the file was updated, cost data were entered into new statements, and the calculator was used to check the cost data and to determine the total ARC project costs.

02/27/70 140610

JUMP TO SUCCESSOR

7075 INASAI COSTS 1960

Period	Salary	Burden	Orhd	NonLabr	TotCost	Fee	TotChgs
1	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-
3	8402	1407	9443	301	10032	1200	21217
4	10201	2004	12243	707	24355	1400	26004
5	11027	2260	13287	2002	30031	1570	31601
6	10220	1564	11023	1469	20280	1402	20010
7	12354	2359	10140	1277	20104	1920	22104
8	11201	2143	12000	12522	20453	2401	19814
9	9430	1717	10225	2450	23720	1514	25244
10	9472	1399	10020	1244	22359	1492	24047
11	11137	2119	12744	1023	20047	1920	21947
12	10005	1916	11521	6002	29525	1048	27611
13	10200	1930	11774	8220	32104	2042	34449

FIGURE II-1 A BRANCH OF FILE HISCO

02/27/70 140710

JUMP TO ITER

7101 INASAI COSTS 1960

Period	Salary	Burden	Orhd	NonLabr	TotCost	Fee	TotChgs
1	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-
3	670	120	722	23002	24639	739	25370
4	1000	209	1234	10041	20390	417	21190
5	1010	153	1160	2076	26245	707	27492
6	1107	221	1334	25039	27377	971	28344
7	2499	674	2054	24434	28341	507	28843
8	2522	479	2001	25002	30005	224	21011
9	2032	507	2532	25413	28474	1700	27900
10	4707	909	2460	25270	27142	1114	28257
11	2734	520	2124	10140	14550	1394	16944
12	2303	420	2620	4702	14147	429	14592
13	4466	810	2102	67147	77542	2214	72090

TA-7101-14

FIGURE II-2 A BRANCH OF FILE HISCO

001
0
000

JUMP TO SUCCESSOR
1

02/27/70 1400:10

7101 (BARC) COSTS 1969

Period	Salary	Garden	Gnd	Variable	TotCost	Fee	TotChgs
1	9110	1770	10446	10000	49760	1492	51332
2	3006	1500	6001	7000	21000	600	21600
3	5006	2000	9100	11000	31000	1100	32100
4	7007	1000	1500	13000	31000	1500	32500
5	7000	2000	15000	13000	40000	1000	41000
6	9000	2000	15000	13000	40000	1000	41000
7	10000	2000	15000	13000	40000	1000	41000
8	11000	2000	15000	13000	40000	1000	41000
9	12000	2000	15000	13000	40000	1000	41000
10	13000	2000	15000	13000	40000	1000	41000
11	14000	2000	15000	13000	40000	1000	41000
12	15000	2000	15000	13000	40000	1000	41000
13	16000	2000	15000	13000	40000	1000	41000
pre(jcum	170000	15000	200000	130000	1100000	10000	1110000

1

FIGURE II-3 A BRANCH OF FILE HISCO

001
0
000

JUMP TO IZEN
1

02/27/70 1400:10

Combined Project Salary 1969

Periods	7101	7013	* Total
1			
2			
3	600	0000	9100
4	1000	10000	11000
5	1010	11000	12010
6	1100	10000	11100
7	1000	10000	11000
8	1000	10000	11000
9	1000	10000	11000
10	1000	10000	11000
11	1000	10000	11000
12	1000	10000	11000
13	1000	10000	11000
14	1000	10000	11000
15	1000	10000	11000
16	1000	10000	11000
17	1000	10000	11000
18	1000	10000	11000
19	1000	10000	11000
20	1000	10000	11000
21	1000	10000	11000
22	1000	10000	11000
23	1000	10000	11000
24	1000	10000	11000
25	1000	10000	11000
26	1000	10000	11000
27	1000	10000	11000
28	1000	10000	11000
29	1000	10000	11000
30	1000	10000	11000
31	1000	10000	11000
32	1000	10000	11000
33	1000	10000	11000
34	1000	10000	11000
35	1000	10000	11000
36	1000	10000	11000
37	1000	10000	11000
38	1000	10000	11000
39	1000	10000	11000
40	1000	10000	11000
41	1000	10000	11000
42	1000	10000	11000
43	1000	10000	11000
44	1000	10000	11000
45	1000	10000	11000
46	1000	10000	11000
47	1000	10000	11000
48	1000	10000	11000
49	1000	10000	11000
50	1000	10000	11000
51	1000	10000	11000
52	1000	10000	11000
53	1000	10000	11000
54	1000	10000	11000
55	1000	10000	11000
56	1000	10000	11000
57	1000	10000	11000
58	1000	10000	11000
59	1000	10000	11000
60	1000	10000	11000
61	1000	10000	11000
62	1000	10000	11000
63	1000	10000	11000
64	1000	10000	11000
65	1000	10000	11000
66	1000	10000	11000
67	1000	10000	11000
68	1000	10000	11000
69	1000	10000	11000
70	1000	10000	11000
71	1000	10000	11000
72	1000	10000	11000
73	1000	10000	11000
74	1000	10000	11000
75	1000	10000	11000
76	1000	10000	11000
77	1000	10000	11000
78	1000	10000	11000
79	1000	10000	11000
80	1000	10000	11000
81	1000	10000	11000
82	1000	10000	11000
83	1000	10000	11000
84	1000	10000	11000
85	1000	10000	11000
86	1000	10000	11000
87	1000	10000	11000
88	1000	10000	11000
89	1000	10000	11000
90	1000	10000	11000
91	1000	10000	11000
92	1000	10000	11000
93	1000	10000	11000
94	1000	10000	11000
95	1000	10000	11000
96	1000	10000	11000
97	1000	10000	11000
98	1000	10000	11000
99	1000	10000	11000
100	1000	10000	11000
101	1000	10000	11000
102	1000	10000	11000
103	1000	10000	11000
104	1000	10000	11000
105	1000	10000	11000
106	1000	10000	11000
107	1000	10000	11000
108	1000	10000	11000
109	1000	10000	11000
110	1000	10000	11000
111	1000	10000	11000
112	1000	10000	11000
113	1000	10000	11000
114	1000	10000	11000
115	1000	10000	11000
116	1000	10000	11000
117	1000	10000	11000
118	1000	10000	11000
119	1000	10000	11000
120	1000	10000	11000
121	1000	10000	11000
122	1000	10000	11000
123	1000	10000	11000
124	1000	10000	11000
125	1000	10000	11000
126	1000	10000	11000
127	1000	10000	11000
128	1000	10000	11000
129	1000	10000	11000
130	1000	10000	11000
131	1000	10000	11000
132	1000	10000	11000
133	1000	10000	11000
134	1000	10000	11000
135	1000	10000	11000
136	1000	10000	11000
137	1000	10000	11000
138	1000	10000	11000
139	1000	10000	11000
140	1000	10000	11000
141	1000	10000	11000
142	1000	10000	11000
143	1000	10000	11000
144	1000	10000	11000
145	1000	10000	11000
146	1000	10000	11000
147	1000	10000	11000
148	1000	10000	11000
149	1000	10000	11000
150	1000	10000	11000
151	1000	10000	11000
152	1000	10000	11000
153	1000	10000	11000
154	1000	10000	11000
155	1000	10000	11000
156	1000	10000	11000
157	1000	10000	11000
158	1000	10000	11000
159	1000	10000	11000
160	1000	10000	11000
161	1000	10000	11000
162	1000	10000	11000
163	1000	10000	11000
164	1000	10000	11000
165	1000	10000	11000
166	1000	10000	11000
167	1000	10000	11000
168	1000	10000	11000
169	1000	10000	11000
170	1000	10000	11000
171	1000	10000	11000
172	1000	10000	11000
173	1000	10000	11000
174	1000	10000	11000
175	1000	10000	11000
176	1000	10000	11000
177	1000	10000	11000
178	1000	10000	11000
179	1000	10000	11000
180	1000	10000	11000
181	1000	10000	11000
182	1000	10000	11000
183	1000	10000	11000
184	1000	10000	11000
185	1000	10000	11000
186	1000	10000	11000
187	1000	10000	11000
188	1000	10000	11000
189	1000	10000	11000
190	1000	10000	11000
191	1000	10000	11000
192	1000	10000	11000
193	1000	10000	11000
194	1000	10000	11000
195	1000	10000	11000
196	1000	10000	11000
197	1000	10000	11000
198	1000	10000	11000
199	1000	10000	11000
200	1000	10000	11000
201	1000	10000	11000
202	1000	10000	11000
203	1000	10000	11000
204	1000	10000	11000
205	1000	10000	11000
206	1000	10000	11000
207	1000	10000	11000
208	1000	10000	11000
209	1000	10000	11000
210	1000	10000	11000
211	1000	10000	11000
212	1000	10000	11000
213	1000	10000	11000
214	1000	10000	11000
215	1000	10000	11000
216	1000	10000	11000
217	1000	10000	11000
218	1000	10000	11000
219	1000	10000	11000
220	1000	10000	11000
221	1000	10000	11000
222	1000	10000	11000
223	1000	10000	11000
224	1000	10000	11000
225	1000	10000	11000
226	1000	10000	11000
227	1000	10000	11000
228	1000	10000	11000
229	1000	10000	11000
230	1000	10000	11000
231	1000	10000	11000
232	1000	10000	11000
233	1000	10000	11000
234	1000	10000	11000
235	1000	10000	11000
236	1000	10000	11000
237	1000	10000	11000
238	1000	10000	11000
239	1000	10000	11000
240	1000	10000	11000
241	1000	10000	11000
242	1000	10000	11000
243	1000	10000	11000
244	1000	10000	11000
245	1000	10000	11000
246	1000	10000	11000
247	1000	10000	11000
248	1000	10000	11000
249	1000	10000	11000
250	1000	10000	11000
251	1000	10000	11000
252	1000	10000	11000
253	1000	10000	11000
254	1000	10000	11000
255	1000	10000	11000
256	1000	10000	11000
257	1000	10000	11000
258	1000	10000	11000
259	1000	10000	11000
260	1000	10000	11000
261	1000	10000	11000
262	1000	10000	11000
263	1000	10000	11000
264	1000	10000	11000
265	1000	10000	11000
266	1000	10000	11000
267	1000	10000	11000
268	1000	10000	11000
269	1000	10000	11000
270	1000	10000	11000
271	1000	10000	11000
272	1000	10000	11000
273	1000	10000	11000
274	1000	10000	11000
275	1000	10000	11000
276	1000	10000	11000
277	1000	10000	11000
278	1000	10000	11000
279	1000	10000	11000
280	1000	10000	11000
281	1000	10000	11000
282	1000	10000	11000
283	1000	10000	11000
284	1000	10000	11000
285	1000	10000	11000
286	1000	10000	11000
287	1000	10000	11000
288	1000	10000	11000
289	1000	10000	11000
290	1000	10000	11000
291	1000	10000	11000
292	1000	10000	11000
293	1000	10000	11000
294	1000	10000	11000
295	1000	10000	11000
296	1000	10000	11000
297	1000	10000	11000
298	1000	10000	11000
299	1000	10000	11000
300	1000	10000	11000
301	1000	10000	11000
302	1000	10000	11000
303	1000	10000	11000

Sec. II MANAGEMENT SYSTEM

This employed the ADD, SUBTRACT, MULTIPLY and DIVIDE capabilities and used the four holding registers.

The calculator package has an 'INSERT' command that inserts the current contents of the calculator's accumulator into the file text as indicated by a bug selection. Work with HISCO indicated that a 'replace' command would be very desirable.

The usual way of accessing HISCO was via pre-established links from other working files whenever the user had a question about recent costs. The VIEWSPECs in the link usually caused HISCO to be brought in with only high-level statements on display, showing only the headings for project name, combined salary, total charges, and total ARC costs (see Fig. II-6).

The user could then select the project he was interested in (by the command JUMP TO ITEM) open up an additional level for viewing, and see column headings and numerical data (Figs. II-1, II-2, and II-3).

Then he could jump down through the accounting periods to the one he was looking for.

If he was making a calculation (perhaps already started in the file he was working in before he linked to HISCO), he could then call the calculator and add, subtract, multiply or divide by any of the numbers in HISCO. His previous calculations while in the previous file would remain intact.

If finished with HISCO, he could then return to the previous file (by the command JUMP TO FILE RETURN) and continue with the calculation, having found in HISCO the input number or numbers he was looking for.

Such a sequence occurs very fast. Experience with HISCO seems to prove the value of having a simple calculator built into NLS, where it is instantly available when needed and can interact directly with data in an NLS file.

Desk calculators are available for most people who need to do basic arithmetic work, but when one is looking through extensive files for inputs to calculations, the conventional calculator is not

nearly as useful as this on-line version.

Summary: As an arena for experimentation, HISCO proved very valuable. Operationally, it was useful from time to time but revealed a need for more frequent updating of the summary data. Our experience with HISCO led to the development of a redesigned cost-history file called COSTS.

File COSTS

This file is updated weekly, with 4-week and cumulative summaries.

The COSTS file is referred to frequently, because the weekly inputs now show trends with considerable sensitivity.

We decided that the elements most useful to us for this year are the following:

- (a) Salary costs
- (b) Total personnel costs
- (c) Non-labor costs
- (d) Total costs
- (e) Total charges with fee
- (f) Balance remaining

See Figs. II-7, II-8, and II-9. Figures II-7 and II-8 show the same branch of the file with different VIEWSPECs; Fig. II-8 displays one more level than Fig. II-7, and this level shows the weekly data. Figure II-9 shows the weekly data for another project.

We also decided to include funding information showing current totals, unfunded totals, and total contract amounts in the categories cost, fee, and total.

We use separate branches for each project and for total ARC project costs (Fig. II-10). The skeleton format for the file was set up in advance for the entire year of 1970.

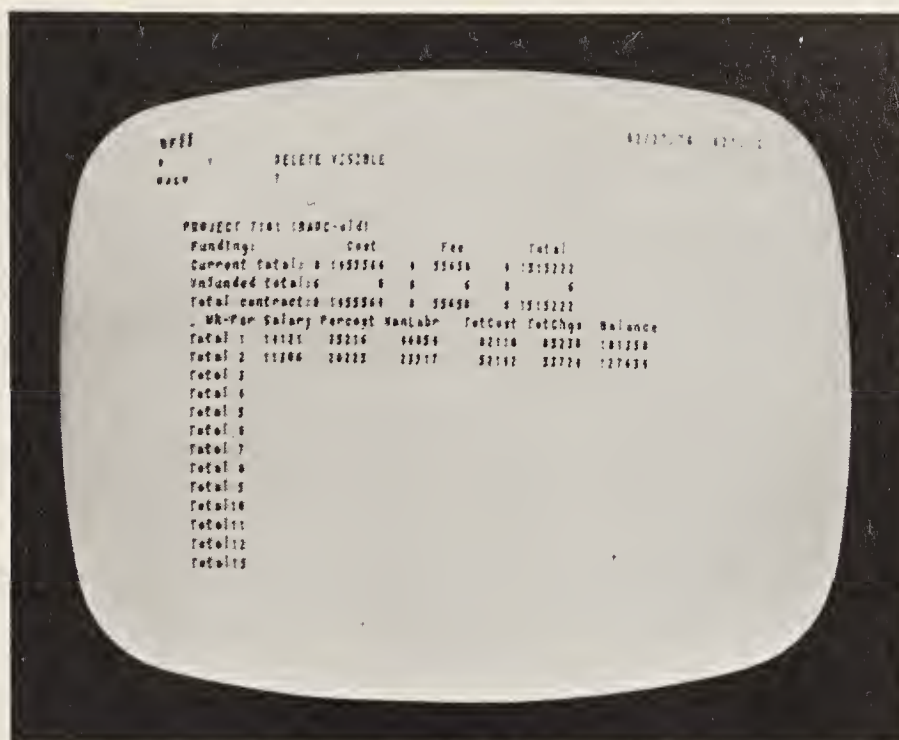


FIGURE II-7 A BRANCH OF FILE COSTS, SHOWING ENTRIES FOR 4-WEEK ACCOUNTING PERIODS



TA-7101-17

FIGURE II-8 SAME AS FIGURE II-7, BUT EXPANDED TO SHOW WEEKLY ENTRIES



FIGURE II-9 SAME AS FIGURE II-8, BUT FOR A DIFFERENT BRANCH OF FILE COSTS SHOWING DATA FOR A DIFFERENT PROJECT



FIGURE II-10 A BRANCH OF FILE COSTS SHOWING COMBINED DATA FOR ALL ARC PROJECTS

Our approach was to create a separate statement for each week, one level below the "total" statements for each 4-week period. For the second week of 1970 (which is in the first accounting period) the statement starts with a 2-1 and then, proceeding across the line, shows the amounts listed above in six columns (Figs. II-8 and II-9).

Before entering any actual data, the first top-level branch (containing some 70 statements) was copied within the file at the same level four or five times. Then each blank branch simply had the project name headings inserted for the project using that branch. We keep one extra blank format branch available in case any new projects should arrive.

Like HISCO, COSTS is usually reached through a link from some other working file, perhaps while a study of near-future costs is in progress, or from an ongoing proposal cost estimate. Again the file is usually entered with only the top-level statements or project headings showing (see Fig. II-11).

If a particular project is of interest, that branch is selected and another level opened for view. The second level shows period-by-period subtotals in each cost category (Fig. II-7). If weekly data are desired, another level is opened by changing the VIEWSPECS (Fig. II-6) and a particular week is selected by the command JUMP TO ITEM.

The statement for each week has the week ending date as its name. The reason for this is not only so that the statement for a particular week can be accessed by the JUMP TO NAME command using the ending date, but also so that the date may optionally be suppressed from the display. NLS has the capability of suppressing all statement names from the display.

The normal way of looking at the file is with names suppressed; thus the dates do not clutter the display; however, a user who needs to know the ending date for a particular week can see it by executing a single command.

To access the information for another project within COSTS, one executes JUMP TO RETURN twice to see the

top-level statements again (Fig. II-11).

One can move very quickly and accurately through a file that is set up in this fashion, even without any familiarity with the information it contains.

The primary function of COSTS is to show a consistent week-by-week progression of costs for each project by category. The file can also be used for study purposes, through the use of content-analyzer patterns, some of which are stored in the header statement (see Fig. II-12, which is the same as Fig. II-11 but with different VIEWSPECS). Any other patterns can be created as needed.

This allows a user to extract special categories of information from the file very quickly. For example, a user may easily create a display showing all project costs for the eighth week of 1970, for each ARC project. It is also possible to output such a "filtered" display via a line printer, thus obtaining hard copy of a special-purpose extract from the total file.

The content analyzer is helpful when using the calculator on all the data for one week, project by project, to find total ARC charges by category.

When only one week's data are displayed, one can add items down each column and insert the answer in the "ARC total" space. One can then clear the accumulator, and add down the next column. This is done very rapidly through bug selection of input numbers and keyset entry of commands -- ADD, ADD, ADD, ADD, INSERT, CLEAR, ADD, ADD, ADD, ADD, INSERT, CLEAR, and so forth.

Figures II-13 and II-14 are before/after photos of this process.

The COSTS file is now operationally useful to us, and we expect it to be useful for future experimentation with automatic processing techniques.

b. Estimates

Proposals

Another use of the system is in creating proposal cost

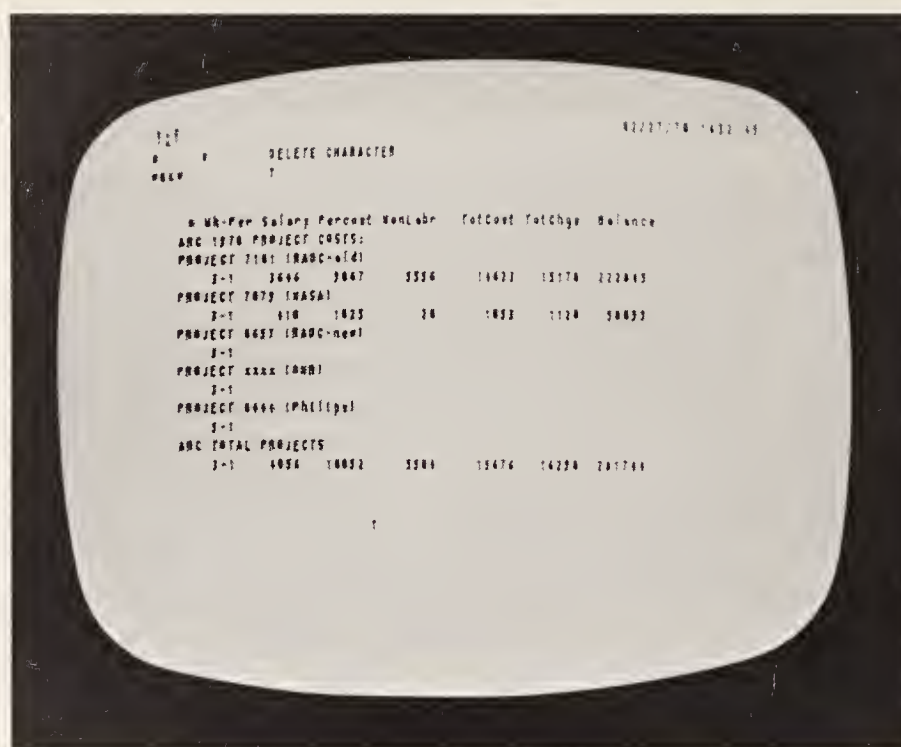
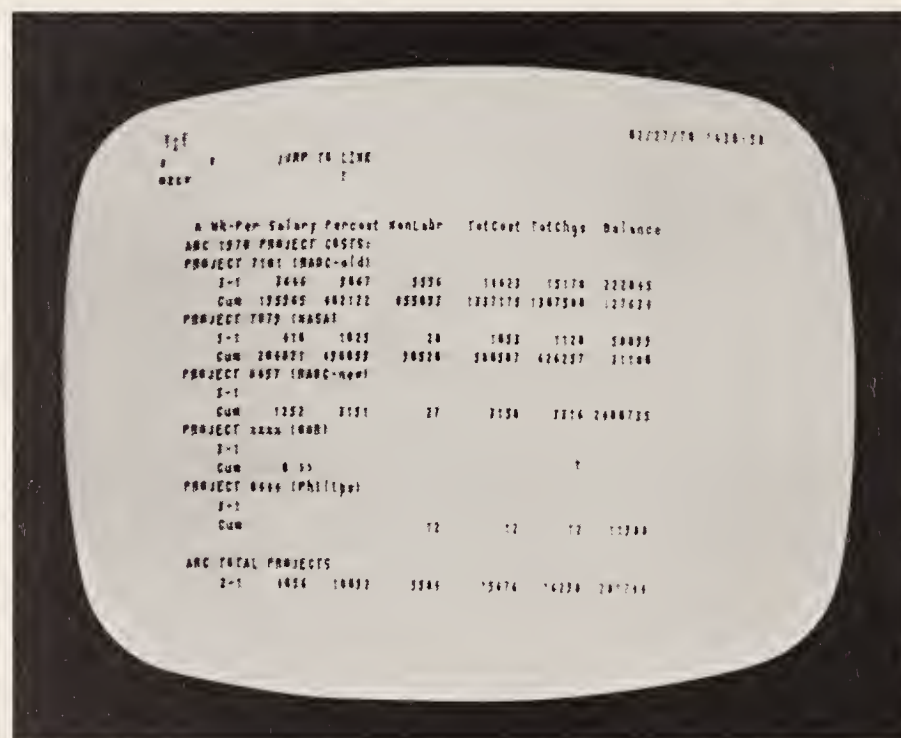


FIGURE II-13 VIEW OF FILE COSTS WITH CONTENT ANALYZER IN OPERATION, SHOWING DATA FOR ONLY A SINGLE WEEK. This is done by using the first pattern appearing in square brackets in FIGURE II-12.



TA-7101-20

FIGURE II-14 SAME AS FIGURE II-13, BUT AFTER A USER HAS INSERTED CUMULATIVE TOTALS IN THE COLUMNS

estimates. We first estimate the amount of effort required for the proposed work. To estimate the cost of this effort, we make reference to various on-line files. The estimating process typically proceeds along the following lines.

Personnel Costs

The estimator loads a special file, maintained by himself, which is a directory to all of his other files and perhaps to a few files belonging to other people. Figures II-15 and II-16 are two displays of a user's file directory. In Fig. II-15, only first-level statements are shown; these are used for establishing categories. In Fig. II-16, another level is shown, containing the actual directory listings in each category.

This "file directory" contains links to each of the files that it lists. In the present case the files probably would be cost histories, personnel listings, previous special studies of costs, and other administrative information.

He loads a previous cost estimate, makes a working copy of it, changes the heading to reflect the name of the new proposal estimate, and eliminates the amounts from the old estimate.

This produces a blank cost estimate format. If any items from the old estimate are inappropriate, they are easily deleted; new items are easily added as separate statements. When the format is ready, it is output as a new file.

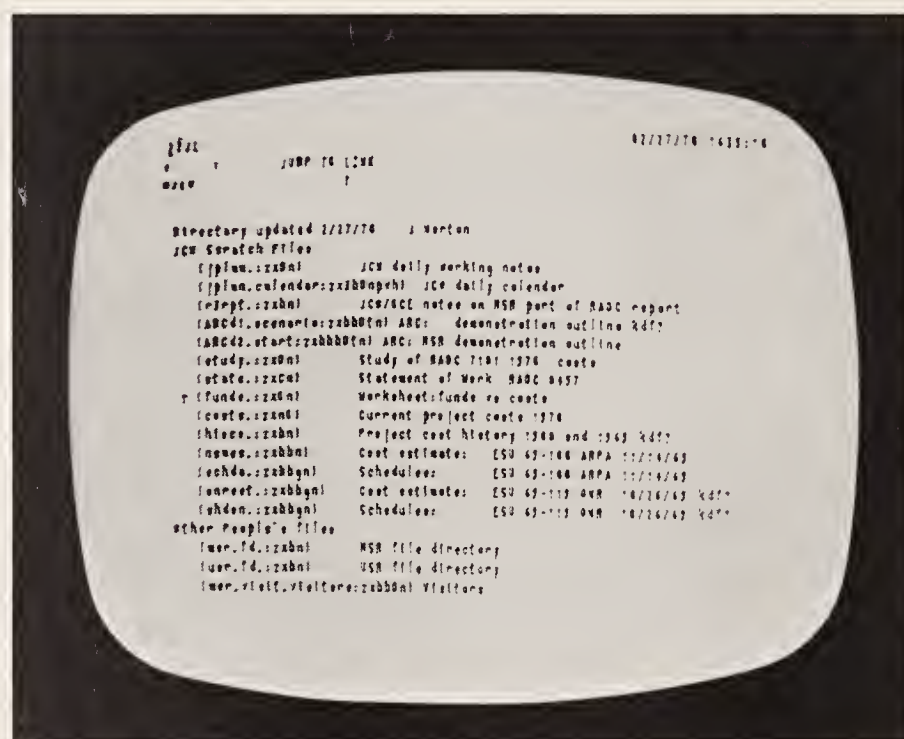
He can then load a file that lists names of people in the group and some projection of expected additions. Figures II-17, II-18, and II-19 show portions of such a file.

Using this personnel-listing file, he obtains information about labor categories. A branch containing content-analyzer patterns is kept in the file. These can be easily reached by jumping to a link which causes all the patterns to be displayed (Fig. II-20).

Each pattern will select some particular



FIGURE II-15 VIEW OF A USER'S FILE DIRECTORY, SHOWING FIRST-LEVEL STATEMENTS ONLY

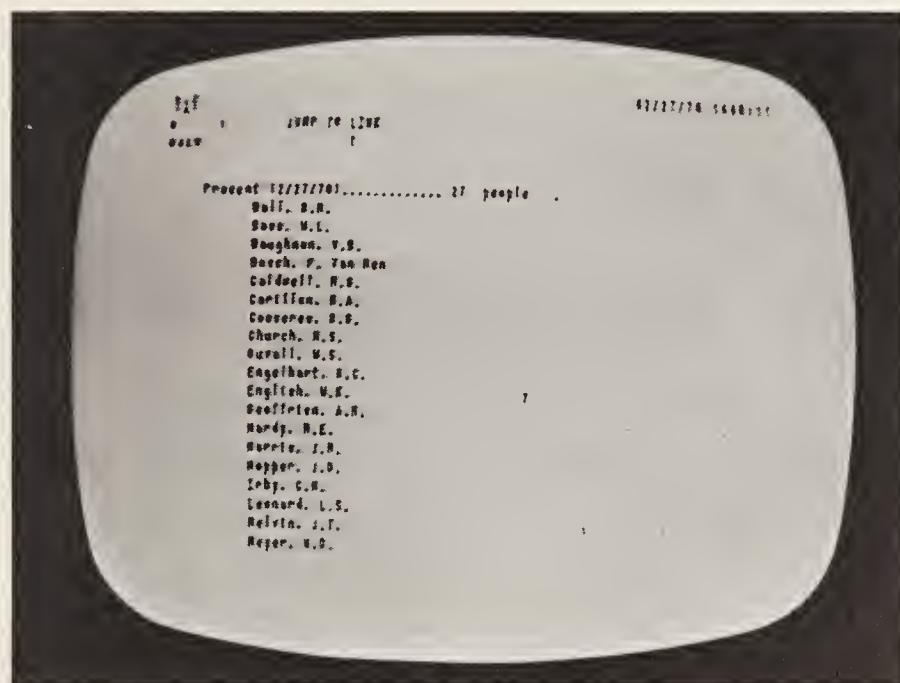


TA-7101-21

FIGURE II-16 SAME AS FIGURE II-15, BUT WITH ALL LEVELS DISPLAYED



FIGURE II-17 PART OF A FILE CONTAINING INFORMATION ON ARC PERSONNEL. Not all levels are shown.



TA-7101-22

FIGURE II-18 A VIEW OBTAINED BY JUMPING TO ONE OF THE STATEMENTS SHOWN IN FIGURE II-17 AND OPENING AN ADDITIONAL LEVEL

category of statements from the file. For example, the estimator will need to know which people have the status of Senior Professional.

He selects the appropriate pattern with the command EXECUTE CONTENT ANALYZER, and then jumps on a link which turns on the content analyzer, starting the search at the beginning of the branch containing personnel listings and restricting the search to that branch.

This produces a display showing only the listing of senior professionals in the group. This set of statements can then be transferred to the new proposal cost estimate file.

Other patterns can be used to extract sets of statements according to other criteria -- for example, all the hardware or software people in the group (Figs. II-21 and II-22).

Thus the estimator can select, by labor category, representative people who may be involved with the proposal; as he selects them, he can transfer their names and the information that goes with them to the file where he is building up his estimate.

At present we do not keep individual salary information on line, although we could do this if we added some security measures. Calculations for the average salary category, based on the specific people contemplated, are made off-line at present.

These average salary amounts are inserted into the on-line cost estimate. The calculator is used to multiply numbers of man-months times average salaries per month to determine total salary costs per labor category and overall direct labor totals. All of this is achieved within the actual file that will become the finished estimate.

The payroll burden and overhead rates are checked for currency and inserted into the estimate, using the calculator to apply them to the direct labor. At this point the labor portion of the estimate is completed.

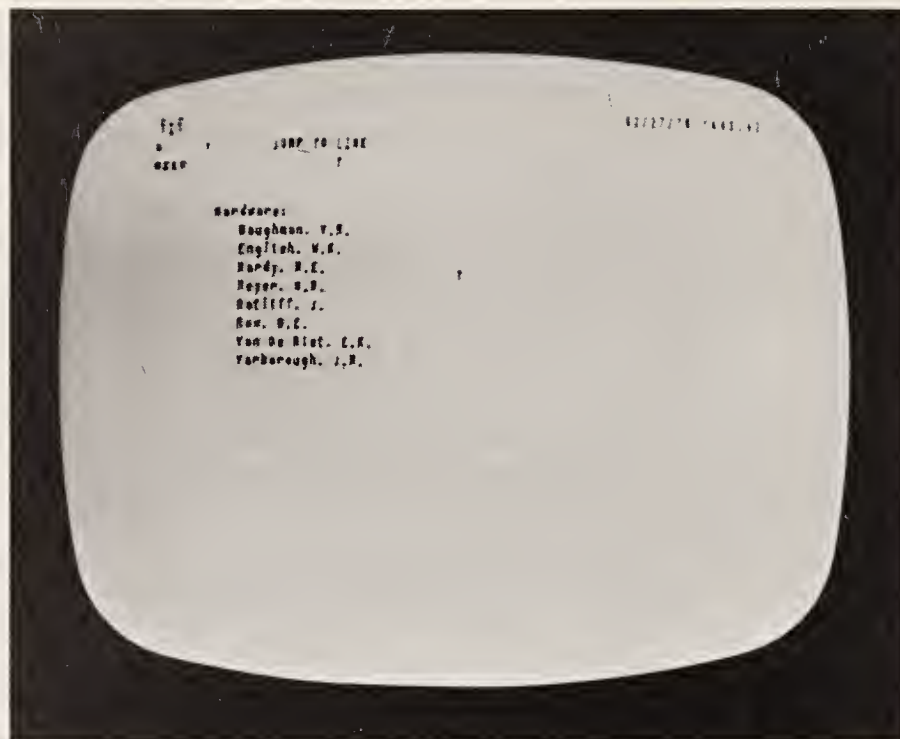


FIGURE II-21 VIEW OBTAINED BY USING CONTENT ANALYZER TO SELECT ENTRIES IN PERSONNEL-INFORMATION FILE THAT ARE TAGGED FOR "HARDWARE"

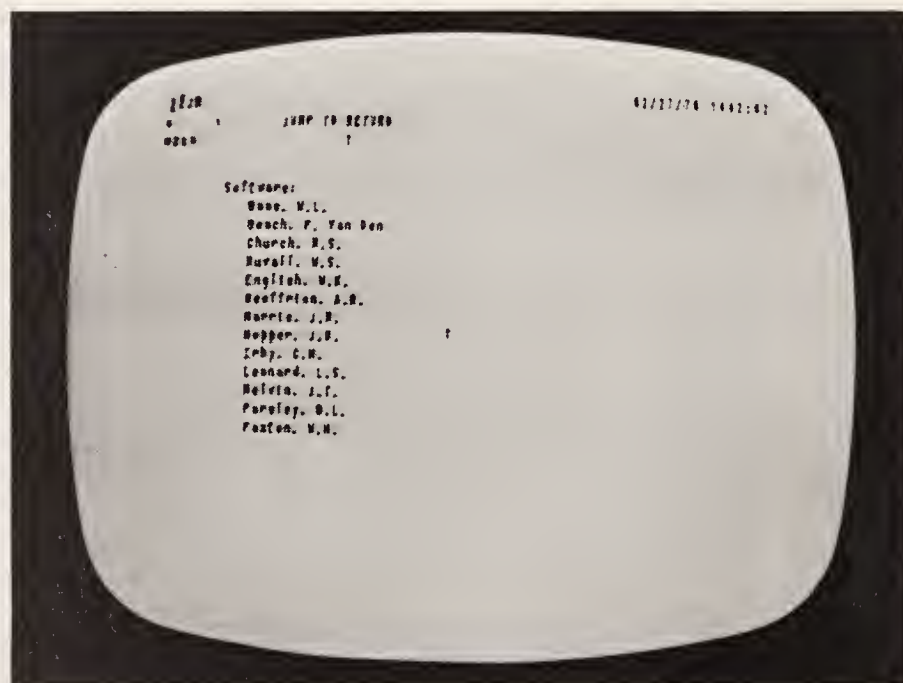


FIGURE II-22 VIEW OBTAINED BY USING CONTENT ANALYZER TO SELECT ENTRIES IN PERSONNEL-INFORMATION FILE THAT ARE TAGGED FOR "SOFTWARE"

Non-Labor Costs

A typical estimate will involve some travel costs, some consultant costs, and some report costs. Data supporting the cost of consultants may be checked by reviewing current consultants' costs by project and by consultant. These are kept in a separate file and reached through a link for review. The data may be copied into the estimate if some of the information is of use.

Report production costs are estimated using current Institute schedules, which are based primarily on the number of pages expected in the end product. These computations can be made using the calculator, and the existing cost factors from the last proposal, checked for current applicability.

In addition, there may be plans to add equipment in the proposal. In this case, the estimator will use an equipment study written in another file by the people involved in hardware design.

The equipment costs contained in the special study are summarized in total and reached by a link. The special study can be viewed and updated as appropriate and can be copied to go with the proposal as an appendix or used later for back up.

In this fashion, various information is gathered from various files and transferred into the developing cost estimate. Figures II-23, II-24, and II-25 show various portions of a completed on-line cost estimate as actually used for a recent ARC proposal.

Working Forecasts

Operational Use of Estimates

As the project progresses, proposals and estimates can also be used as guides for management of the project. It is useful to forecast the expected project costs on either a four-week period or monthly basis.

This can be done by creating a new file using the type of format that the COSTS file uses. We insert total figures from the cost estimate, using the calculator to determine average rates and specific estimated

```

TLE
0 1 JUMP TO LINE
0000

***** 02/02/70 0000:50 JCM 1,015:01,RT1:01,0PB:01,ME0:01 SRI Proposal
Isaker (funds:annual:zzb001) (schdo:1:zzb001)

COST ESTIMATE
for the two year period starting 1/0/701
Personnel Costs 0 1,212,500
Direct Costs 1,033,179
Total Estimated Cost 2,245,679
Fixed Fee 115,334
Total Estimated Cost Plus Fixed Fee 2,361,013
* See attached Schedule

Personnel assumed:

```

FIGURE II-23 PART OF AN ON-LINE COST ESTIMATE FOR USE IN A PROPOSAL

```

TLE
000 000 JUMP TO LINE
0000

Direct Costs 1,033,179
Travel 7,300
10 trips Cost = 100 = 5,000
10 Days Subsistence = 15 = 1,000
Auto Rental 500
Facility 0
Consultants 10,000
Report Costs 1,027
Total Direct Costs 1,033,179

Total Estimated Cost 2,245,679
Fixed Fee 115,334
Total Estimated Cost Plus Fixed Fee 2,361,013

* See attached Schedule
.SEG:

```

TA-7101-25

FIGURE II-24 PART OF AN ON-LINE COST ESTIMATE FOR USE IN A PROPOSAL

JR27
a
0200

JUMP TO RETURN
1

02/27/70 1444.13

FACILITY COSTS

Total Facility Cost:	\$ 1,044,672
Computer Facility Support	652,672
Lease Cost	\$ 431,372
Maintenance and Operation	21,300
Anticipated Improvements	191,000
Interactive External Core	62,000
Conferencing Facility	13,200
Console Switching Facility	67,000
Univac drum interface	12,000
Bryant disc expansion	20,000
Disc Interface IBM 72421	10,000

FIGURE II-25 PART OF AN ON-LINE COST ESTIMATE FOR USE IN A PROPOSAL

vr28
a
0200

FREEZE STATEMENT
1

02/02/70 1520:09

40 Stand. Typewriter, 32P. New Elling Corp., 1 at 624.52, 20130, PR A00925, February 17, 1970, dec.

42 Stripper, Thermal Wire, Model TW-1, and Glade, Techni-Tool, Inc., 1 each at 691.99 (total), PR A00927, 710961, February 18, 1970, chr.

44 Handrol replacement (wul), 65551-7, ABP, Inc., 1 at 647.03, 710961, PR A00927, February 18, 1970, chr.

45 Unwrap Tool, A71070 470, Wrap Tool, 14H-10, Slocum, 10000 08, 910, A26365-422, Gardner-Denver, 710961, February PR A00921 and PR A00920, February 18, 1970, chr.

48 Table, Steelmaster, 7407, 1 at 606.65, 6457-20, PR A00926, February 17, 1970, chr.

41 Cert. Scope-Pablo, 200-1, Tektronix Inc., 1 at 605.00, PR A00911, 20130, February 6, 1970, chr.

TA-7101-26

FIGURE II-26 VIEW OF A PORTION OF THE PURCHASE-ORDER PROCESSING FILE, SHOWING CONTENTS OF INDIVIDUAL STATEMENTS

amounts, and insert answers into the file as it builds. This month-by-month estimate can be reached through a link from working cost files, from the original estimate, or any other file where the question of monthly estimated project costs may arise.

c. Purchase-Order Processing

In making an estimate of costs for new equipment being constructed at ARC, reference to previous cost information is very useful. We have constructed a purchase-order/requisition processing file which contains a separate statement for each item purchased for the past two years at ARC. Figure II-26 shows a portion of this file.

Each statement contains the following information about each purchase:

(1) Total price

This is entered as the statement name.

At present this is not used as an NLS name, but as a way of eliminating information from the screen at will, keeping a consistent location in columnar form for such totals.

(2) Description of item

(3) Vendor

(4) Number of units purchased and price per unit

(5) Purchase Requisition number

(6) Date requisition sent

(7) Purchase Order number when order is placed

(8) Date order is placed

(9) Project or account charged

(10) Date order is received

(11) When the order is completed, it is marked with the special code *comp*. This can be detected by a content-analyzer pattern.

All outstanding orders are contained at a second level under a single branch (see Fig. II-27); therefore the distinction between outstanding and completed orders is easy to see just by reference to level. To reduce clerical error, we consider an order completed when the *comp* pattern is inserted and the statement is moved to its alphabetical position on the top level.

This file can be searched using the content analyzer in some interesting ways. We can ask for all items purchased from a particular vendor on any particular project and see only those. If we wonder about the unit price of a thermal wire stripper, model 2W-1, we can quickly get that information. If we wonder what we purchased on PR A08927, that comes simply by executing a content analyzer pattern specifying the number. We can see all outstanding orders charged to a particular project quickly. Figure II-28 shows a content-analyzer pattern that has been temporarily written into the file, for finding any entries pertaining to orders for relays under Project 7101. Figure II-29 shows a view generated by using this pattern.

This file is useful, then, from a project-administration standpoint, from the standpoint of following a purchase requisition from the order stage through completion, and also for providing backup information for cost estimates.

This file can also be used as a tickler file by inserting a pattern in the "outstanding requisitions" branch which shows the date we feel we should follow up on the order. Each day one can ask for all those items that have the current date as a follow-up date.

This file is kept up-to-date by the secretary of the hardware group, who is most involved with requisitioning. She does this updating entirely with TODAS.

d. Summary on the Systematic Use of Project Cost Files

One by one each of these files might be interesting. As a combination, quickly available to many users, their utility seems remarkable.

A cost study, as discussed above, can rely on all previous project costs as recorded in the system and can draw on those files for inputs. One can draw on the personnel roster file by labor category, work interest or as extended into a skills inventory.

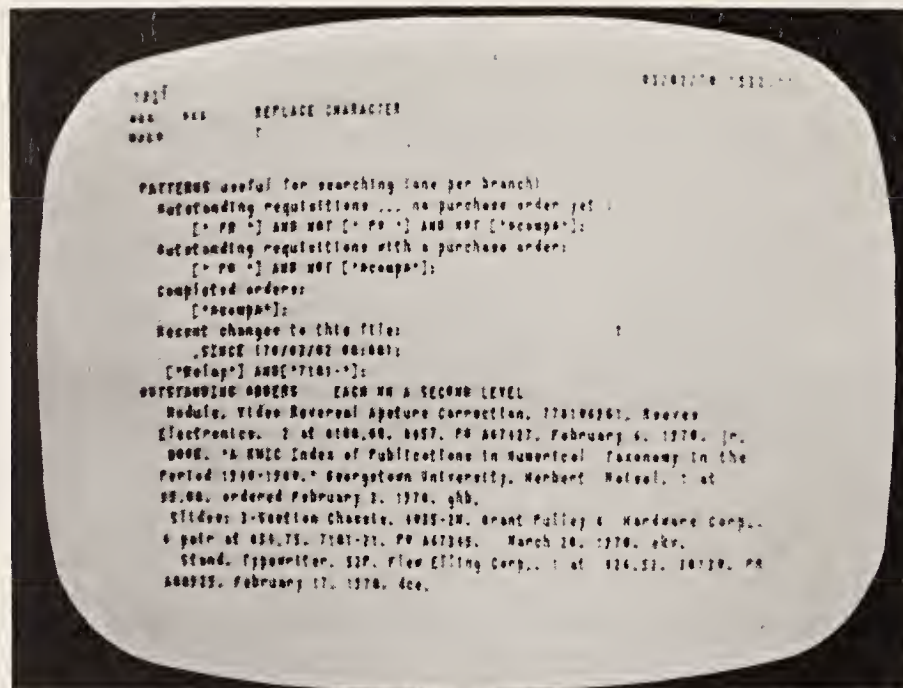
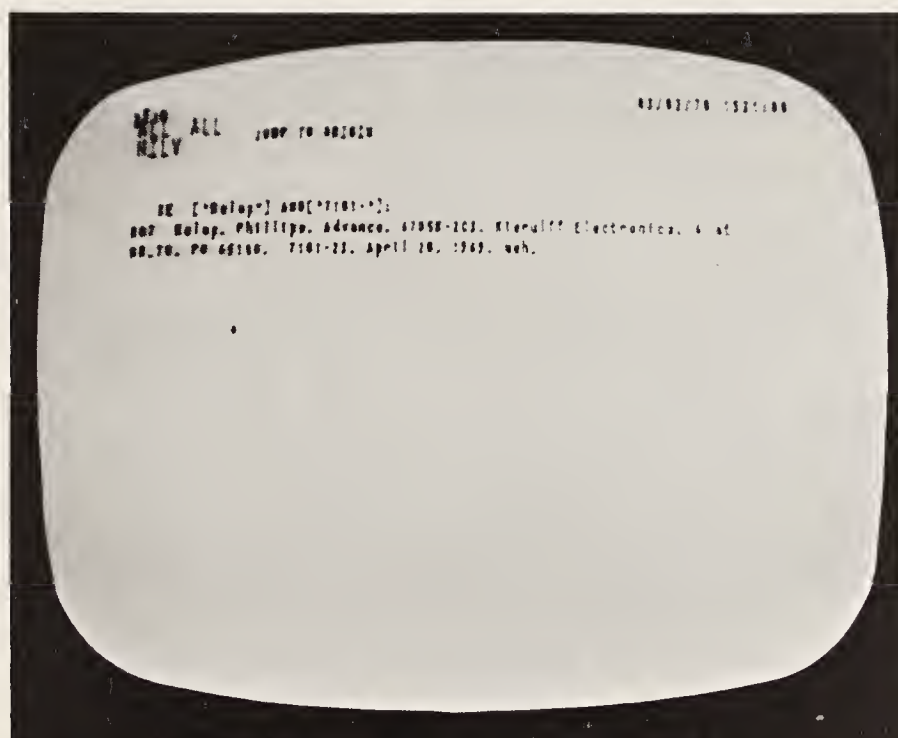


FIGURE II-27 VIEW OF A PORTION OF THE PURCHASE-ORDER PROCESSING FILE, SHOWING OUTSTANDING ORDERS LOCATED IN A SEPARATE BRANCH—UPPER PART OF SCREEN SHOWS A BRANCH CONTAINING CONTENT-ANALYZER PATTERNS



TA-7101-27

FIGURE II-28 A CONTENT-ANALYZER PATTERN FOR SEARCHING IN THE PURCHASE-ORDER FILE



TA-7101-28

FIGURE II-29 VIEW GENERATED BY A SEARCH ON THE PATTERN SHOWN IN FIGURE II-28

We can browse through the purchase-order file, reflecting the current or previous costs per item. We can link to activity-planning files to see which people are involved with various ongoing tasks and to see on what tasks we are contemplating certain equipment purchases. We can link to proposal cost estimates for month-by-month cost projections.

These files can be accessed in any order, from any direction, at any time, with only a few keystrokes by the user. They are also accessible remotely through the use of TODAS, thereby giving mobility to the user with less load on the system.

Our main objective in making cost studies is to arrive at solid sets of projections or other answers as quickly and effectively as possible. Direct on-line access to input information is extremely helpful.

3. Activity Planning and Status

a. Introduction

Section II-B-2 describes the experimental establishment of TODAS Development Activity and discusses its method of operation. One facet of TODAS work is the extensive experimental use of on-line files as aids in conducting meetings and formulating plans. This section gives some details on the construction and use of these files.

b. Planning and Status Files for TODAS Development Activity

File UPLAN

The planning file for the TODAS Development activity contains a branch with comments on how to use the file, a branch for content-analyzer patterns, and a branch containing actual task plans.

The task-planning branch has, as substatements, task categories which include documentation plans, teaching plans, design plans, META plans, and inactive task plans. The levels under these categories contain separate task plans, such as "TODAS REFERENCE GUIDE DEVELOPMENT," "USER EXPERIMENTS RELATED TO TODAS," and "TEXT MANIPULATION SYSTEMS BIBLIOGRAPHY."

Each task branch contains comments by the task

leader on the following:

- (1) Description of the task, with links to other working files used in its development
- (2) Comments on the relationship of the task to other ARC tasks
- (3) Estimates of people involved (with levels of effort and timing)
- (4) Status comments

UPLAN is linked to from another file called UMEET (described below), which is used for on-line note-taking during meetings of the TODAS group. Portions of UPLAN can be temporarily copied into UMEET for use during meetings.

UPLAN contains a blank task format in a separate branch. Whenever a new task is added, this branch is copied into the appropriate planning area (such as documentation plans). Then the name of the task is inserted as a heading along with the initials of the task leader.

Certain items in this file are useful in content-analysis searches. The most useful are the initials of people involved in tasks, the milestones, the estimates, and the status. To make content-analysis searches more consistent, asterisks are placed before such items.

With an appropriate pattern, one can then ask a question such as "what is the involvement of a particular person in this activity?" task by task. All branches with estimates containing the specified initials and an asterisk will then be shown. The same branches show expected levels of effort.

Since this is the only information displayed on the screen, it is relatively easy to see potential conflicts in the allocation of a person's time between tasks for this activity or to make a hard copy of this displayed information on the line printer.

The content analyzer can also return statements commenting on the status of tasks, so that a quick survey of all such comments can be made. This is particularly useful for coordination of several tasks and for

preparing for meetings of the group.

When many people try to update the same file, serious problems are created. This is a well-known situation (discussed further in Appendix B). If two people are both working on the file, one person's work may be lost when someone else who has been using the file writes his copy back out on the disc. Therefore we tried to introduce a convention where people place a signal of some sort in the file when it is in use.

This procedure was not well used, probably because people were generally in too much of a hurry. Therefore, some work was lost. We found that it was easier, with the present file-handling limitations, have research assistants do the updating on the file gathering information from various people as needed.

Part of the description for a task involves the specification of significant milestones, if possible. The task leader has to have some idea of important milestones during the progress of the work and must develop some feeling for whether these milestones are occurring within the resources expected to be allocated to the task.

We tried an on-line task-planning chart, showing 10-week periods where milestones could be marked for each task. Milestones were indicated by showing an NLS name for each milestone statement (see Fig. II-30). Therefore, viewing this task-planning chart on a display, we could "JUMP TO NAME", selecting one of the milestone points on the chart, and a description of the milestone and its relationship to the task would then be displayed. A "JUMP TO RETURN" brought back the planning chart.

This shows some promise of being useful in the future, but some refinements in display technique and milestone selection are necessary before it can become operational.

Another use of the content analyzer is to search for entries made "since or before" a certain date, or for entries made by certain people. This makes it easy to see who has been updating the file recently, and what they have done to it.

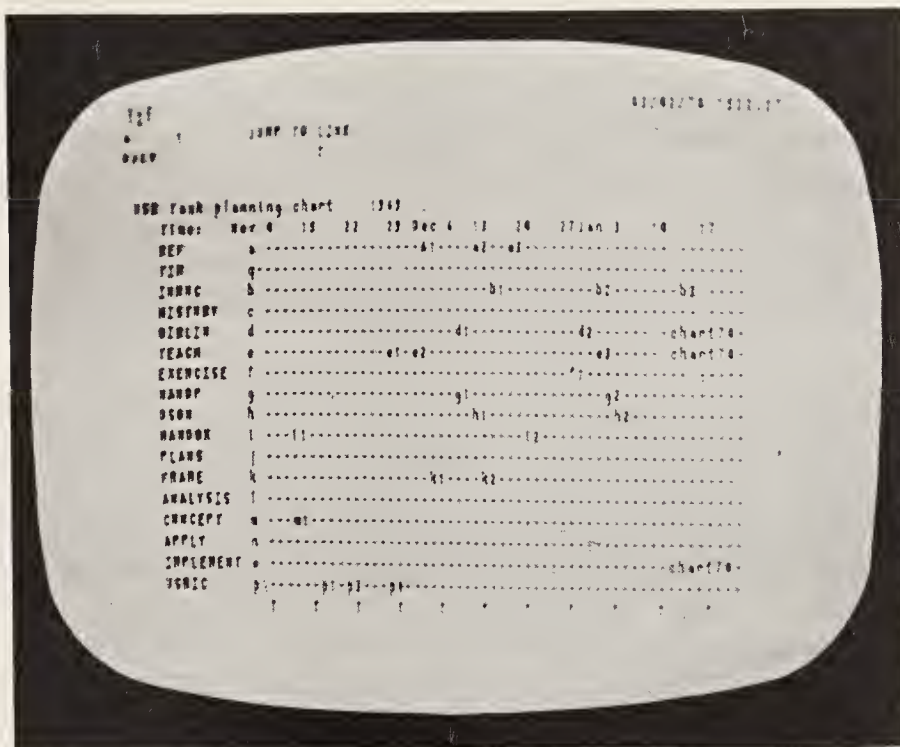
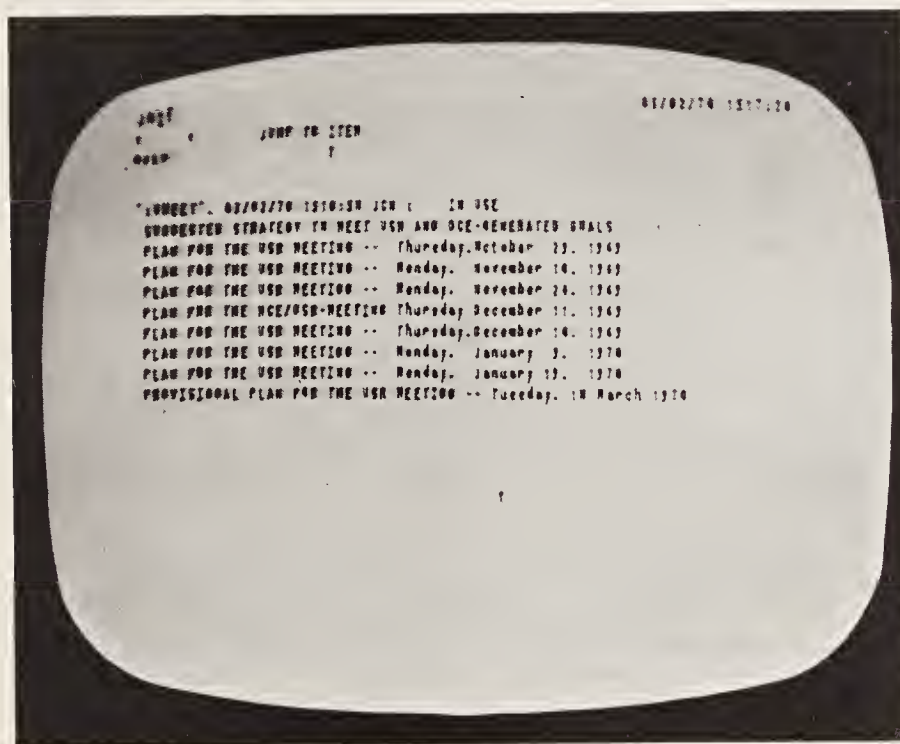


FIGURE II-30 TASK MILESTONE CHART FROM FILE UPLAN



TA-7101-29

FIGURE II-31 TOP-LEVEL VIEW OF FILE UMEET, SHOWING ACCUMULATION OF NOTES FROM A SERIES OF MEETINGS IN A SINGLE FILE

This is of less importance for a person who is updating his own file, for he probably remembers the kinds of things he has changed. When many people work on the same file, it is helpful to know who has been changing it and in what areas they have been working.

File UMEET

We created a separate file called UMEET for plans and notes from the TODAS activity meetings.

This file is similar to the UPLAN file in format. On-line note-taking by a research assistant, as practiced in the user system and software groups, has proven quite useful for recording important parts of discussions during meetings. The on-line note taker has not been a distracting influence in meetings; in fact, she has contributed at times. She is available for finding information in the file and for recording special ideas in other files upon request during the meetings.

Meetings are conducted with hard-copy agenda distributed before each meeting. The on-line notetaker has an on-line version of the same agenda in front of her. As the discussion proceeds, she makes her notes right in the on-line agenda.

Items left for discussion in following meetings, or as special questions to be resolved before the next meeting, can be marked by the note-taker and retrieved from the file for later study.

When the meeting is completed, the notes are condensed to a meaningful summary, distributed to the participants, and displayed on a bulletin board. In other words, the agenda for a particular meeting is developed, during the meeting, into minutes of the meeting. A copy of the unaltered agenda is also kept.

Successive meeting agenda and minutes are kept in one file (see Fig. II-31). This permits us to search for discussions of various topics and to receive answers in chronological order.

B. Organization Studies

Our organizational studies have centered on two topics. The first of these is the study of the "On-Line Community" -- our own ARC group seen as a unique example of a small, close community of workers who make intensive use of on-line computer aids in their daily work.

The second area of concentration has been the implementation of two experiments on organization structure and planning methods in such a community.

1. On-Line Community

Our study of the On-Line Community is described here in terms of the total working environment of the group and the structuring of staff roles within the group.

a. Environment

We consider the total working environment, for purposes of this study, to consist of the physical environment and the "user environment." The latter is a general term intended to indicate the existence, availability, and performance of the numerous on-line aids used by the group.

Physical Environment

We have changed the basic work room or laboratory configuration from isolated one-man offices and a remote shop and computer/work room to one-man offices opening directly onto an open, courtyard-like work area. We still use a remote shop and computer room due to building layout restrictions. The consoles were moved out of the offices into this central working area. We have put in separate lighting circuits so we can turn off lights in different parts of the room, reducing reflections on the displays. Within the work area, the consoles can easily be regrouped to permit users to work cooperatively.

One effect of this was to change the personal interaction pattern dramatically, simply by increasing the amount of interaction.

A second effect was to permit much more effective utilization of the display facility; the facility is much more "available" than it otherwise would have been.

Within the general work area, the consoles (which are of several different designs offering different advantages) are set up in varying configurations, with differing arrangements for lighting, seating, proximity to other consoles, etc. In general, the individual configurations can be quickly and flexibly altered as various needs arise. As a result, an individual who is about to start a working session at a console has a considerable choice of conditions. Figure II-32 shows four views of console in the work area, in actual use for various modes of work.

A further modification to the physical environment was the addition of light movable partitions, for visual privacy. These are low enough so that a person, when sitting, does not see other people working but can, by standing or moving his chair two or three feet, contact or 5 other people working at consoles. Most people apparently prefer to partition off only the front of their work stations. Partitions are rarely moved into positions completely surrounding the work stations. When seclusion is wanted, people tend to work in the Herman Miller experimental office, which is isolated from the general work area by high partitions.

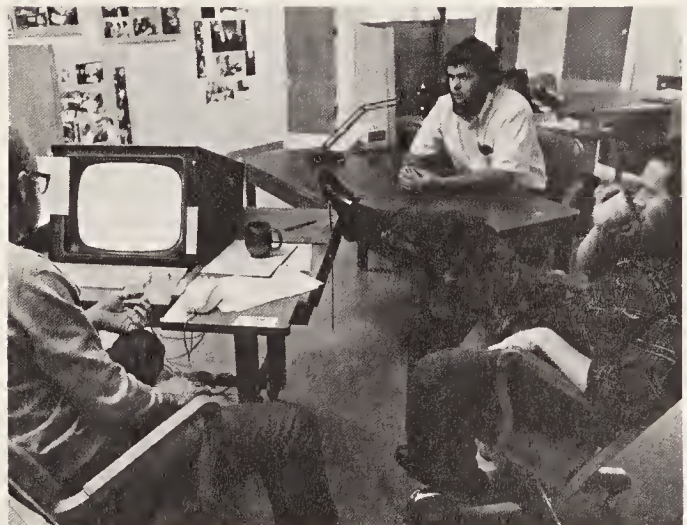
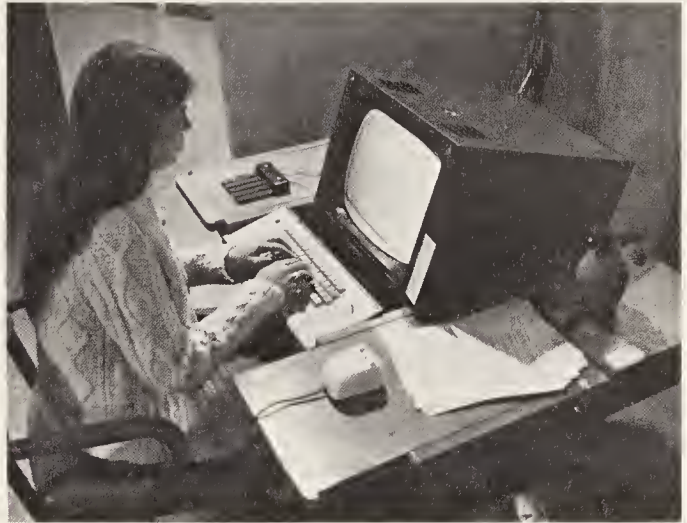
The Herman Miller office has also become the place where the system is demonstrated to visitors. Visitors have the feeling that they are inside the working environment, and no one else is bothered by the visitors' presence.

We have adopted the practice of holding some types of meetings in the Herman Miller area around one or two displays, with a research assistant taking on-line notes

We have found that display viewing is difficult, and multiple-participant access to the system ineffective with meetings of more than three or four people.

On the basis of our experiences with such meetings, we are now redesigning the conference facility (see Sec. II-C-2-a).

We have found that it is highly desirable to make use of the system both night and day. Night access to our work area is inconvenienced to some extent by the existing security measures, particularly when we wish to work with



TA-7101-31

FIGURE II-32 VIEWS OF CONSOLES IN USE IN THE ARC WORK AREA

non-SRI personnel, such as consultants. A much more open and accessible working environment would be greatly preferred.

We see great practical utility in having a maximally flexible physical environment. Each time we have increased the flexibility of the environment, work interaction has increased without any damaging increase in social interaction.

User Environment

During these two years we have provided a useful, though still evolving, on-line text editing and file manipulation system, NLS. This system provides new tool for personal and group use. Appendix A describes NLS in considerable detail from a user's point of view. Appendix D is a technical description of NLS.

We have also developed the Typewriter-Oriented Documentation-Aid System, TODAS (see Appendix A). This provides some of the same features as NLS but can be used remotely by people not physically in the facility. TODAS will produce considerably less load on the timesharing system than NLS. We have experimented with remote use of TODAS using portable typewriter terminals with acoustic couplers. The resulting mobility, with direct access to all of our files, shows interesting possibilities for team collaboration, together or physically remote.

With the introduction of TODAS, we have provided more opportunity for people to interact with the ARC files from their offices, although some of the processes are slower. There has not yet been widespread use of TODAS, but this will change with improvement in service capacity of the system and addition of new features to TODAS. Availability of several 30-character/second typewriter terminals will also greatly increase the value of TODAS.

b. Staff Functions and Activities Within ARC

Activities we have identified as basic include the following:

- (1) Hardware
- (2) Software

- (3) Management System Research
- (4) User System Research
- (5) ARPA Network Participation
- (6) Operational Management of ARC.

Staff functions for each activity involve the specification, design, implementation, documentation, evaluation, and maintenance process as new system features are added.

As we hire hardware and software people, research assistants, and secretaries, our policy has been that a person's capabilities must go beyond any narrow specialization. A highly skilled systems programmer must have additional background before he can be used effectively in this group.

We need people who are capable of both long- and short-range planning, participating in goal and subgoal setting, and contributing to the the design, implementation, and other processes.

For most ARC work it is important that people be primarily oriented toward designing and building tasks and less toward contemplative and reflective ones. However, since our work mixes both research and development modes we must be capable of acting in either capacity at different stages in the implementation of any given task. It is also a requirement that people have the ability to focus on different levels of the endeavor, alternating modes frequently as the needs arise.

2. Experiments on Internal Activity Structure

We conducted two experiments on the use of augmented methods for planning work. These experiments were conducted with a newly established group, the TODAS development group, and with a well-established, fairly tight-knit group, the software group.

a. TODAS Development Activity Planning

A part of ARC user system research involves the specification, design, implementation, teaching, use, and evaluation of new features being added to TODAS as related to anticipated ARC and ARPA Network needs.

The TODAS planning experiment was initiated along these lines:

We first developed a strategy for use as the group form and for encouraging it to make further plans directed toward ARC and TODAS-related goals. The steps considered necessary for the group were:

- (1) Identify both internally and externally generated goals

- (2) Agree on structure and mode of operation of the TODAS group, with the following features:

- (a) A group representative reporting to the ARC Manager and to external activities

- (b) A team approach to tasks and planning, with one leader for each task

- (c) Investigation of decision techniques.

- (3) Plan tasks for the group and for the individuals in the group (including tasks already in progress, where applicable). We were to do this according to the following outline:

- (a) Build an easily visible collection of task alternatives, to be modified as appropriate after analysis and review.

- (b) Identify and use the skills in the group, securing other needed skills if not available in the group.

- (c) Estimate participants' level of effort and the timing involved, assessing the net effect of the combined plans.

- (4) Meet periodically to review progress, usually every two weeks.

Meetings were intended to be open to interested staff of ARC, with use of an agreed upon format.

Special discussion meetings (and other forms of communication) for "help" when special problem situations arose were also anticipated.

(5) Maintain a TODAS "information center" on-line and off-line. The basic files were the following:

(a) File FD: File Directory for TODAS-oriented links. This file also contains links to TODAS group participants' personal file directories and links to the following files:

(b) File UMEET: Meeting plans and notes

(c) File UPLAN: Task plans and status notes

(6) Communicate status of TODAS work to the ARC Manager and the ARC staff.

Having determined this strategy, appropriate initial participants were contacted and the group was established.

The group started having meetings and developed a meeting strategy that contained the following elements:

(1) A "facilitator," whose role includes the following:

(a) Preparation of the meeting plan, with inputs from the rest of the group

(b) Guidance during the meeting to ensure that all important items are discussed

(c) Providing an orderly way for new or unexpected items to be discussed as appropriate, or deferred.

This role was rotated among the membership of the group from meeting to meeting, depending on the expected agenda subjects.

(2) A "process watcher," whose role involves attention to processes in operation during the meeting. This includes verbal and nonverbal interactions between people, decision processes, etc.

This was done to give the participants added insight about less obvious features of the meeting.

This role was rotated among the membership of the group from meeting to meeting, depending on the expected agenda subjects.

(3) An on-line note taker, whose role includes the following:

- (a) Distribution of the meeting plan and preparation of the meeting notes outline before the meeting
- (b) Careful recording of important discussions and points made during the meeting
- (c) Retrieval of needed information from on-line files during the meeting
- (d) Summarizing the meeting notes and distributing them after the meeting

The role of the on-line note-taker was filled by two research assistants on an alternating basis. This provided flexibility and ensured that an experienced note-taker was available for each meeting. Information gained at these meeting was valuable to the note-takers in their other day-to-day work.

- (4) Regular participants
- (5) Invited specialists
- (6) A meeting plan and agenda
- (7) Relevant documents produced on-line by any member

Distribution of documents was arranged before each meeting. Documents included descriptions of design changes in TODAS, drafts of teaching documents, etc.

- (8) Tentative plan for the following meeting
- (9) An evaluation of the utility of the meeting.

Notes from meetings were kept on an evolutionary basis as separate branches in one file, UMEET, and also in hard copy for distribution to all members and to a bulletin board.

Planning

We made an easily accessible listing of tasks in progress and under consideration, in a separate file called UPLAN (described above in Sec. II-A-3-o), which can be modified by individual task leaders or by research assistants.

This file helped increase the extent to which meetings were used to evaluate and redesign tasks, instead of to report information that would not be changed by group interaction.

It facilitated the exchange of reportorial information outside the meetings, when individuals could give their full attention to the file.

It was also available during meetings for reference or modification.

Another use of the file was to communicate information to people not directly involved in the activity, i.e., the ARC Manager and others in ARC.

Most of the planning dealt with scheduling and patterns for necessary interaction between tasks and task leaders.

The short-term goals appeared firm enough that we chose not to divert our resources to longer-term goals while this activity was starting.

Interaction

Since this group included people who were involved with other ARC activities such as software, the Network Information Center, and Management Science Research (MSR), it explored some interaction between activities.

It also provided an opportunity for the activity members to be involved in a smaller group than the ARC as a whole. This changed the group dynamics considerably.

The process of identifying internally generated goals stimulated exploration of personal needs of the members

of the group to increase solidarity, mutual liking, understanding, respect, and the desire to cooperate.

Although social interaction initiated at early meetings was beneficial in developing a cohesive working group, progress evaluation at various times indicated that it could then be more effectively continued outside of group meetings to allow more focus on the primary group tasks related to TODAS.

b. Software Activity Planning

The software activity is directed toward the design and implementation of new system software features.

Strategy

This was the second experiment, following the initial results of the TODAS experiment described above. In the two years of the contract, the software group has progressively become more integrated into the total ARC functioning and has doubled in size. One result is that more tasks that depend upon each other are being performed concurrently. The need for each member of the software group to be aware of the progress and design modifications of the tasks undertaken by every other member of the group has increased significantly as the size of the group has grown.

Preplanning by the MSR and group management team included those features found to be most useful from the TODAS activity experiment.

It recognized the existence of leadership responsibilities already in effect, and formalized them.

The same meeting format was used as for the TODAS group. We found immediately that there was more interest in task discussion and plan reformulation and less interest in social interaction and group process than in the TODAS group. As a result, changes made in the planning procedure simplified the documentation to include only essential elements needed for communication by the group members. We also went through the process of listing all current and planned tasks in one consistent format in a file called SOFTP. This resulted in a preliminary listing of 30 critical and separate tasks, with truly

distributed task leadership.

Leadership

Leadership was minimal at the group level, and sufficient because of high motivation to complete tasks on schedule. The strongest leadership was at the task level.

This experiment is still in progress. Longer-range goal and task planning, with better integration with other ARC activity planning, are currently being developed.

c. Summary Comments on Planning Experiments

Active community teamwork, warm human relationships, and good work attitudes are necessary for our organization to function effectively. We must encourage and develop feelings of trust and common goal appreciation so that our people can work closely together over a long period of time, with so much of themselves open to view to others and with such interrelated and challenging tasks to be undertaken. We found that the TODAS group benefited from the initial energy spent on interpersonal relationships, although there was eventually more effort applied to these factors than we found useful for task accomplishment. A careful balance between application of social and work-oriented energy is a necessity.

Although the TODAS experiment was not successful in all respects, it was an experiment where the particular people involved stand a better chance of succeeding in a future experiment with a reoriented group.

Software meetings were judged by participants and outside observers as extremely efficient and effective in meeting predetermined goals. While little attention was paid to interpersonal variables, group morale was strengthened by the meeting procedure. Uncertainties in task definition and individual responsibilities were clarified. The feedback was reported to be useful rather than either flattering or critical. This, again, was a chance for the participants to be involved in a smaller group than ARC. This contributed to the higher morale.

We feel that the techniques developed for meeting and task planning and for on-line note-taking will be useful as they evolve in future activity planning. We need to learn more about realizing the potential of improved interpersonal

relationships in ARC, while expending only a reasonable amount of effort in doing so.

3. Observations From Study of On-Line Community

a. Use of Public Files

The use of public files containing the work of many individual people seems to be well accepted by the group.

Far more communication potential exists in this environment than has yet been realized, although some people have started in some interesting ways.

Our need for development of a Dialogue Support System is clear.

Work habits of the on-line community staff also need development so that they can use the power of existing features and information in the system.

Now is the time for further work on methodology and procedures for use of the system, with the continued parallel evolution of the system itself.

b. System Dependence by the Group

As we augment, we find that it seems less desirable to use conventional tools for many tasks.

This is a problem to be resolved for good use of resources and for the purpose of not overlooking appropriate conventional tools where they can still be very effective.

The various ways that information now gets into the system are:

(1) Direct:

(a) On-line NLS or TODAS use by originator:

Entry of new material

Duplication and/or modification of existing information

(b) On-line NLS or TODAS note-taking at discussions

(2) Indirect:

(a) Transcription sources:

Handwritten

External documents

Stenographic dictation

Recordings

Individual use of dictating equipment

Tape recordings of group meetings

(b) Transcription processes:

Direct NLS use

Direct TODAS use

Paper tape

We are working toward a better assessment of which tools are most appropriate for the various tasks to be performed in ARC.

c. Miscellaneous Observations

This is a work-oriented group. Most people work long hours, usually at an intense rate; little time is spent not actually working.

There are many more work opportunities for the group and for most individuals than there are resources -- in terms of both time and funds.

Group and personal work management involves many difficult choices of tasks to be performed, postponed, or dropped.

The group frequently sets goals at higher levels than it is likely to attain.

This is partly because we want the new features that will make the system more powerful; we are users of our own results.

Sometimes, also, we overassess the potential power of the system, forgetting that it still has limitations, particularly in the area of consistently good service levels. This problem is getting a great deal of attention, however.

The interrelatedness of the on-line community tasks makes planning very difficult, but obviously more necessary.

C. Team Augmentation and Dialogue Support

Our efforts in management research have been centered on the attempt to developing a more closely integrated, participatory way of organizing people, efforts, and resources toward specific goals than is provided by classical management theory.

Toward this goal, we are currently focusing our attention on the problem of improving the management of a working system-development team, using our own organization as the subject of experimentation. This involves two facets of augmentation -- namely, individual augmentation and team augmentation.

Individual augmentation is simply our continuing effort to provide ways of improving the working capability of individual members of a team.

Team augmentation involves the development of improved means for coordinating the efforts of individuals and for integrating their individual contributions into coherent team action.

1. Recent Efforts

A portion of our recent MSR effort has been invested in formulating a "team-augmentation" approach. The initial emphasis is strongly oriented toward the means for communicating and collaborating effectively on issues embedded within a complex and evolving problem domain.

An important facet of this approach has been a preliminary study for a "Dialogue Support System" (DSS) -- a special system of coordinated features which could support the communication and integration of collaborative dialogue among team members.

Appendix B is a more detailed discussion of this formulation, as extracted from the PhD thesis of David A. Evans (see Ref. 1).

2. Future Approaches to Team Augmentation

Experimentation with roles, record-keeping conventions, collaboration procedures, decision-making practices, documentation, etc. will be a rich domain for exploratory MSR work.

The following discussion of fast editing and publication, "super-documents," and augmented conferencing gives a view of some features needed for team augmentation.

a. Fast Editing and Publication

Our already fast editing techniques will continue to evolve, and we plan to concentrate early upon automatic production, from our on-line files, of hard copy having a very flexible composition of text, diagrams, tables, equations, footnotes, and indices.

The design of hard-copy formatting conventions must be related directly to the way in which the associated file material can be studied and manipulated on-line.

b. "Super-Documents"

We have been doing research leading to the development and production of very large, very complex documents containing numerous sections whose details are highly interdependent. These documents will be subject to frequent updating. This will involve further work on techniques for creating and using special indices, footnotes, reader-supportive comments, cross-references, etc.

We currently have quite powerful techniques for aiding an individual or a small report-writing team to produce documents of the usual research-report size and complexity. Part of our approach to team augmentation will be the expansion of these techniques to allow for much greater scope and complexity in documents and much more fluid interaction among the team members who create them.

A team tackling a complex system-development project must provide itself with the highest possible visibility over its working environment -- i.e., over the following factors:

Planning: plans, contingency alternatives, resource commitments, status, criticisms

Design: designs, design principles, constraints, estimates, analyses, supportive data, relevant needs and possibilities

Operation: roles, task definitions, assignments, policies, operational procedures and conventions.

We intend to develop and keep up to date a large, detailed, highly cross-referenced and well-indexed "super-document" that contains just such a description of our own project-team activity. Our techniques for facilitating its modification and republication will be under constant evolutionary pressure.

c. Collaborative Use of On-Line File Systems

On-line access by collaborators to each other's files, as provided by a number of today's time-sharing systems, leave much to be desired in supporting effective dialogue.

An effective dialogue-support system is essential to team augmentation. Hand in hand with the "super-document" facility described above must go some such ability as the following:

Any team member at a display console can study swiftly any portion of the super-document's structured files. Our current system is fairly good for this purpose, but not yet adequate for dialogue study.

Whenever he wishes -- as though he were pencil-marking his private draft with marginal comments, underlines, encircled passages, arrows, etc. -- he can introduce "comments" that are freely sprinkled with explicit references to any specific item (e.g. any character, word, graphic entity, or expression) within anybody's prior entry. (Note: the term "comment" is used here and in the following discussion in a very broad sense -- a comment is any entry which in some way points to a previous entry.)

This commenting capability must be managed by the computer so that it does not matter if other people are simultaneously scanning the same material or affixing comments to the same items.

When creating a comment entry, he needs flexible aids and methods for arranging interspersed or concurrent

display of the referenced passages, for designating the explicit entities he wishes to reference, and for suspending operations temporarily while he checks related material.

Conversely, he needs a way of seeing any comments that reference a passage he is inspecting.

Categories might be defined by authorship, date of creation, text content, or assigned membership in predefined categories.

He also needs a great deal of control over this, however; much of the time he will not want to see any comments, or only comments falling into certain categories.

He also needs considerable control over the way the system displays the comments that he wants to see -- in specified portions of the screen, in full-text or condensed form, etc.

He needs the ability to set up "annunciator calls" to various people, or sets of people, to request their special attention (at some level of priority) to a given comment.

All of the interactive-dialogue entries immediately become part of the super-document, imposing a potentially very complex comment network ("network" because comments can refer to comments in indefinite extension).

It will be hard to keep track of the relationships among these comments and the substantive records about which the dialogue is oriented.

Their relationships need never be ambiguous, but consider the problem of trying to study such a structure to determine where we now stand in our developments and discussion, especially when it is the record of a complex system-design process and the interactive dialogue among very active people.

This is about the most difficult central challenge in effectively augmenting a team -- that of developing computer aids, working methods, etc. to allow a skilled person to be highly effective in digesting the content and implications of such a record, and to

develop a substantive next-stage design or plan that integrates the dialogue contributions.

Essentially similar techniques are required to augment any individual's central intellectual capability for synthesizing the next stage of development in a plan or design. To the extent that we are successful with this, we should be able to offer strong guidance for capability augmentation over wide ranges of individual and team activities.

d. Conference Augmentation

There is great potential value in direct augmentation of conferences and meetings. When people are gathered together to consider a proposal or argument, or to collaborate actively on a problem, there are many possibilities for the development of techniques and facilities to make their work more effective.

There is a wide range of possible approaches to conference augmentation.

At one extreme, each participant would be an experienced NLS user and would have his own console; sophisticated facilities would be provided for "linking" the consoles in various ways to augment communication.

At the other extreme, there would be only a single console with a special operator; special techniques for integrating the NLS facility, the operator, and the conference participants into a working system would be needed.

Between these two extremes, a variety of intermediate approaches is possible.

For any of these approaches, a central problem is the development of conference procedures and the organization of on-line information; both procedures and information structures must be developed in such a way as to gain the greatest possible advantage from the computer facility.

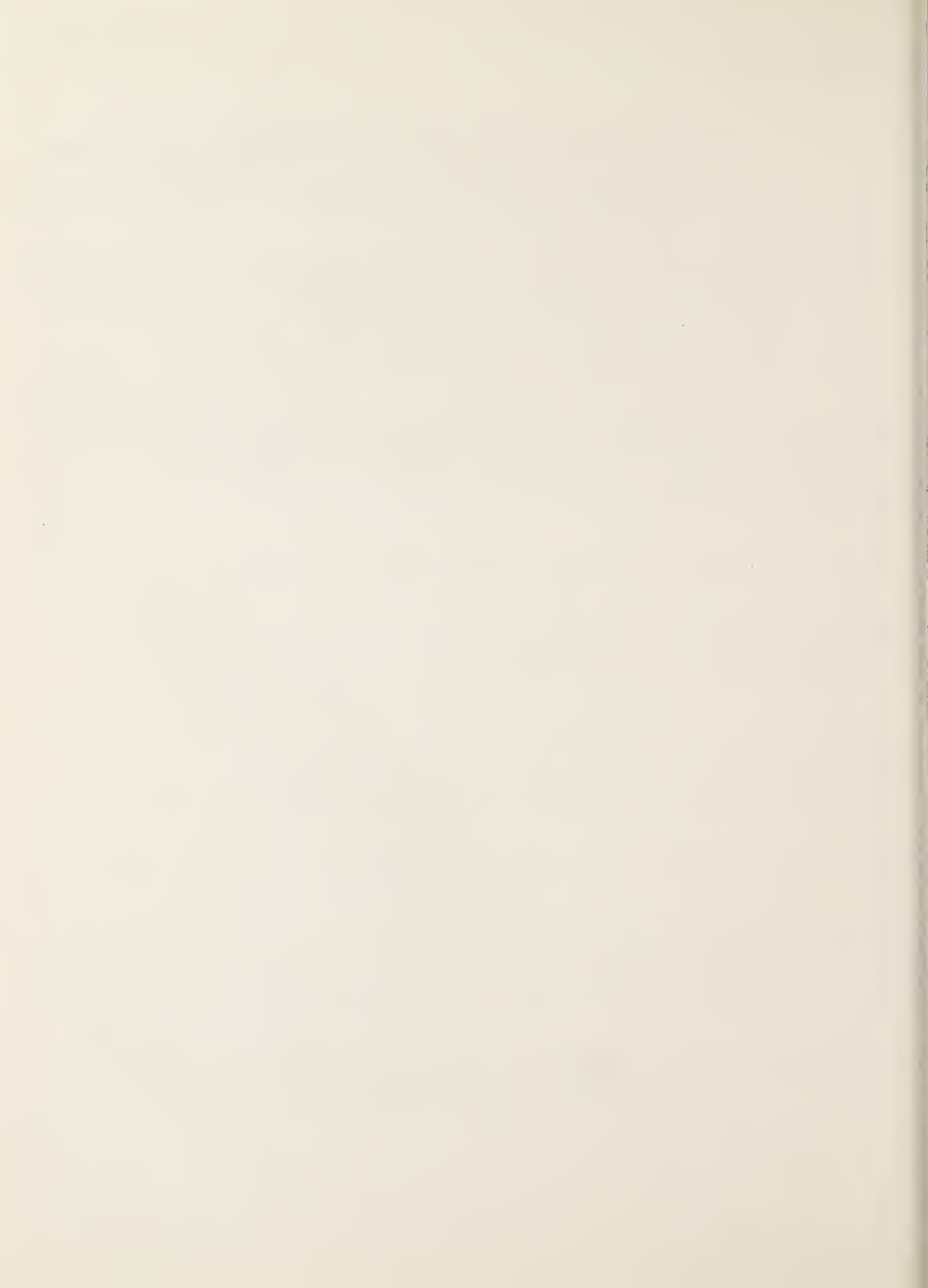
This development of conference procedures and information structures should be done experimentally, under actual usage conditions.

Sec. II MANAGEMENT SYSTEM

We have already experimented with augmenting meetings by having one person operate NLS as an on-line note-taker, where all participants can see the display (see Sec. II-A-3-b).

On the basis of recent experience, we plan to provide better facilities for groups of people working together at consoles and for small meetings where consoles are not available for everyone (or where not all participants are NLS users). This will permit experimentation with intermediate approaches lying between the two extremes described above.

The facility will consist of a meeting room equipped with projection TV, several appropriately designed consoles, and furniture designed so that three or four people may work at the consoles with ten or so less active participants.



A. Introduction

This section reviews the current status of the ARC computer facility and describes the hardware development that has been done during the course of this contract.

The first part briefly describes the computer facility, including both the computer as leased from XDS and the special equipment that has been added by ARC.

The second part discusses modifications and improvements to the facility that have been planned and are now in progress.

The third part presents some comments on features of the system design and discusses some of the reliability and maintenance experience. Because of its unique design, the display system is emphasized. A summary of maintenance costs for the display-generator and television portions of the system is included.

B. The Computer Facility

The configuration of the ARC computer facility has been relatively stable over the past two years. There have been some peripheral additions, in particular the ARPA Network interface and an external core system; these are discussed below.

The current facility is shown in Figs. III-1 and III-2.

1. The Leased Computer

Figure III-1 is a block diagram of the facility as leased from XDS.

A central processor with timesharing hardware operates from a 64K memory in 4 banks with 24-bit words and a cycle time of 1.8 microseconds.

On channels sharing memory access with the CPU are 3 magnetic tape drives, a paper-tape station, and communications equipment for 16 Teletypes.

A second memory buss provides direct access to memory for the RADs (Rapid Access Devices, i.e., drums) and the non-XDS portion of the facility, designated "Special Devices Channel" in Fig. III-1.

There are three drums on the system, operating from a common controller and accessing memory through an XDS device called a Direct Access Communications Channel (DACC). Each drum

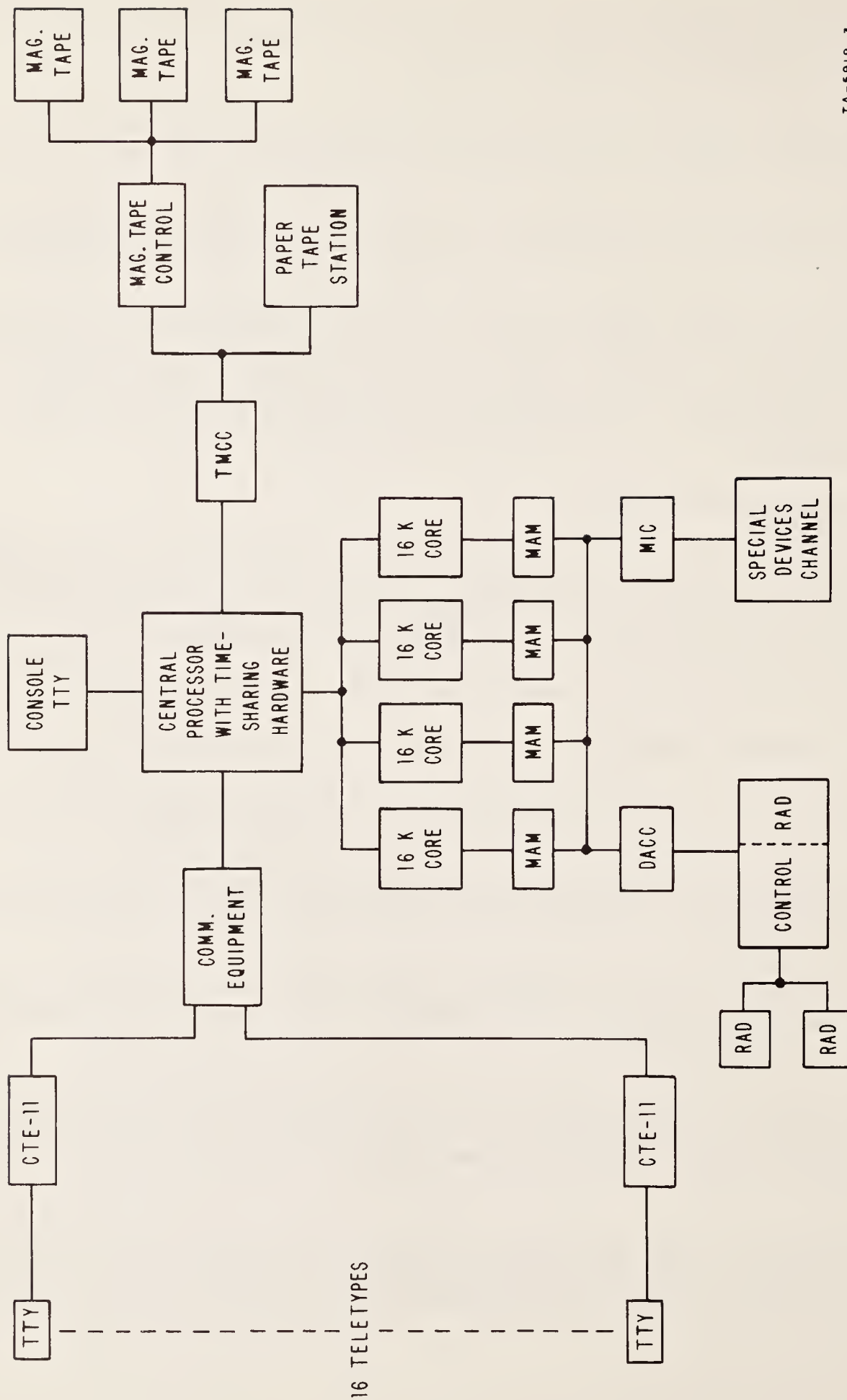
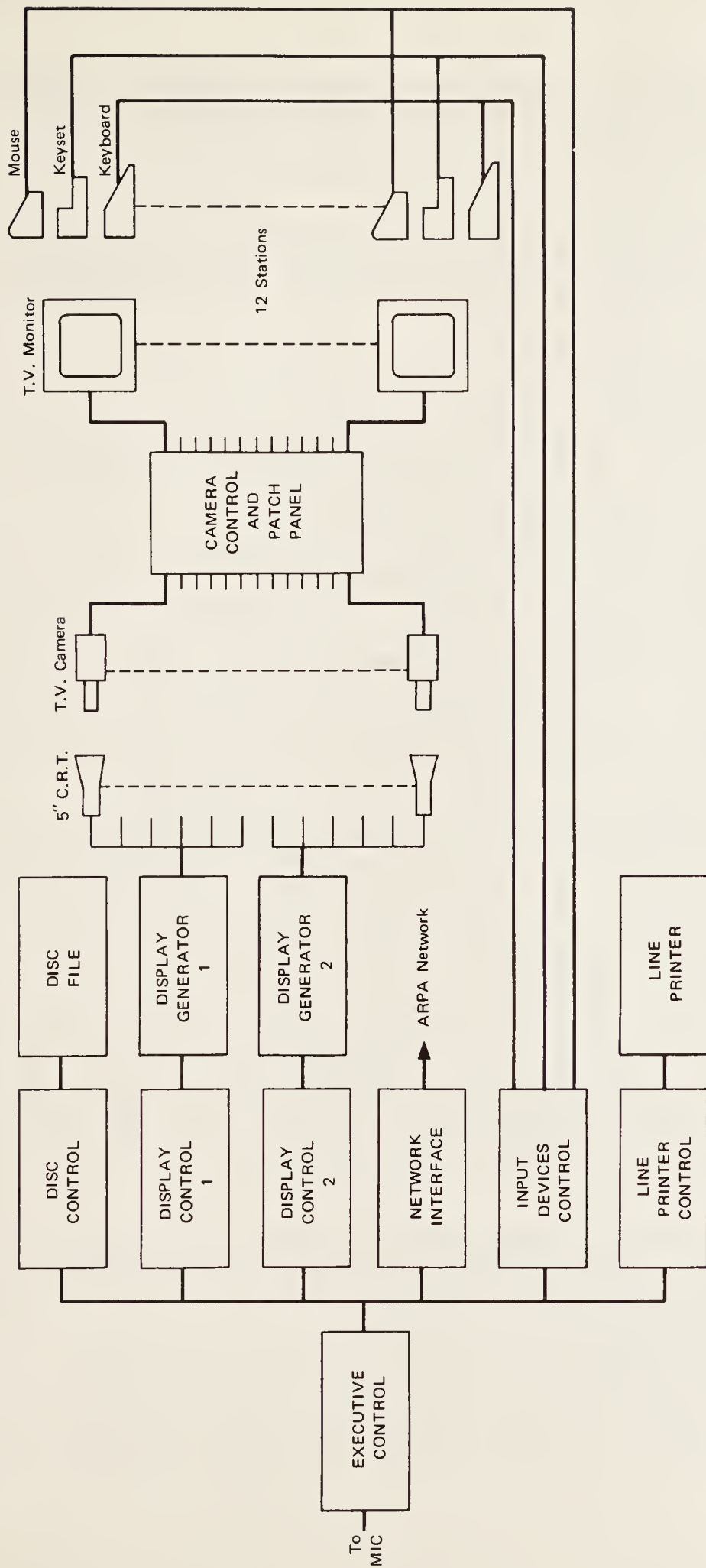


FIGURE III-1 XDS940 COMPUTER FACILITY

TA-5919-3



TA-7101-3

FIGURE III-2 SPECIAL DEVICES CHANNEL

has a capacity of 500,000 24-bit words, a transfer rate of 120,000 words per second, and an average latency of 17 milliseconds.

2. Special Devices Channel

Figure III-2 is a block diagram of the portion of the facility that has been put together by ARC. The following sections describe the major units.

a. Executive Control

The executive control provides an interface to the 940 through the Memory Interface Connection (MIC). It acts as a multiplexer that allows asynchronous access to core by any of the 6 devices connected to it.

The executive control decodes computer input/output instructions and passes them along as signals to the various devices. It accepts interrupts from the devices, synchronizes them, and passes them along to the computer.

It accepts addresses and requests for memory access from the various devices, determines relative priority among them, and synchronizes their access to 940 core.

The executive control includes extensive debugging and monitoring aids. It allows the monitoring of data and addresses for any selected device and permits "off-line" operation of any of the devices.

b. Disc File System

The disc file system consists of a Bryant Model 4061 disc file and associated controller. The system has a capacity of 32 million words, an average access time of 185 milliseconds, and a data transfer rate of 43,000 words per second. A relatively simple field modification will double the present capacity.

The disc controller was designed and built by Bryant to interface with the executive control. Specifications for the controller were developed jointly by Bryant, Project GENIE at UC Berkeley, and SRI.

c. Display System

The display system consists of two identical subsystems,

each with a display controller, a display generator, and 6 high-resolution 5-inch CRTs. A closed-circuit television system carries display images from the CRTs to television monitors in the working area.

The display controllers were designed and built at SRI. They access and process "command tables" that are resident in 940 core.

A command is roughly associated with a user and points to a "display list" in the user's core space. The display list in turn points to buffers containing actual display instructions (commands to the display generator to produce images).

The display controller handles all core accessing, including memory mapping for the user's core space. It passes the display instructions along to the display generator.

The display generators and CRTs were purchased from Tasker Instruments to SRI's specifications. They have general character and vector capabilities.

Presentations for each of the 6 CRTs are generated sequentially, and unblank signals from the display controllers select one or more of the CRTs at a given time.

A high-resolution (875-line) closed-circuit television system transmits display pictures from each CRT to a television monitor at a corresponding work-station console. (Figure II-32 shows several work-station designs.)

d. Input Device Control

In addition to the television monitor, each work station has a keyboard, binary keyset, and mouse. Appendix A describes the use of these devices.

The state of these input devices is read by the input device controller at a preset interval (about 30 milliseconds) and written into a fixed table in 940 core.

Bits are added to information from the keyboards, keysets, and mouse switches to indicate when a new character has been received or when a switch has changed state during the sample period. A new character or

switch change causes an interrupt to be issued at the end of the sample period.

Mouse coordinates are digitized by an A/D converter and formatted by the input device controller as beam-position instructions to the display generator. A user program may include the mouse coordinates, as written by the input device controller, as part of a display list. This allows the mouse position to be continually displayed without attention from the CPU.

e. Line Printer

The line printer is a 96-character drum printer leased from Data Products Corporation (Model M600-11A). With the 96 characters, printing speed is 340 lines per minute.

The line printer controller processes print buffers of arbitrary length (single line buffers are normally used) that have been set up in core by a controlling program. Operation of the printer controller is described in Appendix C.

f. Network Interface

The network interface provides communication between the 940 and an Interface Message Processor (IMP) on the ARPA Computer Network. The interface operates from message buffers in 940 core. Messages to the Network are read by the interface from these buffers and transmitted to the IMP. Similarly, messages received from the IMP are written into buffer space in 940 core. Instructions from the 940 enable the system for receiving messages and control the sending of messages. A "linked-buffer" scheme permits flexible memory allocation.

Operation of the network interface is described in more detail in Appendix C. The interface message processor and its communications protocol are discussed in detail in Ref 2.

C. Modifications in Progress

Two modifications to the facility that will provide significant improvement in service are now being implemented. These are an external core system and faster drums. In addition, an accurate clock system is being added.

1. External Core System

The external core system has been completed and will be integrated into the facility in the near future.

The primary purpose of this core system is to provide storage for display regeneration. Display buffers are presently in "frozen pages" in 940 core -- a significant factor in limiting system response, since they take up space that could otherwise be used for swapping. (See Sec. IV for a discussion of factors affecting response.)

Figure III-3 shows the special devices channel as it will be reconfigured when the core system is integrated.

The inter-core controller controls transfer of data between external core and 940 core. It has two modes of operation:

- (1) A block transfer mode allows the transfer of blocks of up to 2048 words between any two locations in the two cores. (Note that transfer can be between two locations in the same core.)

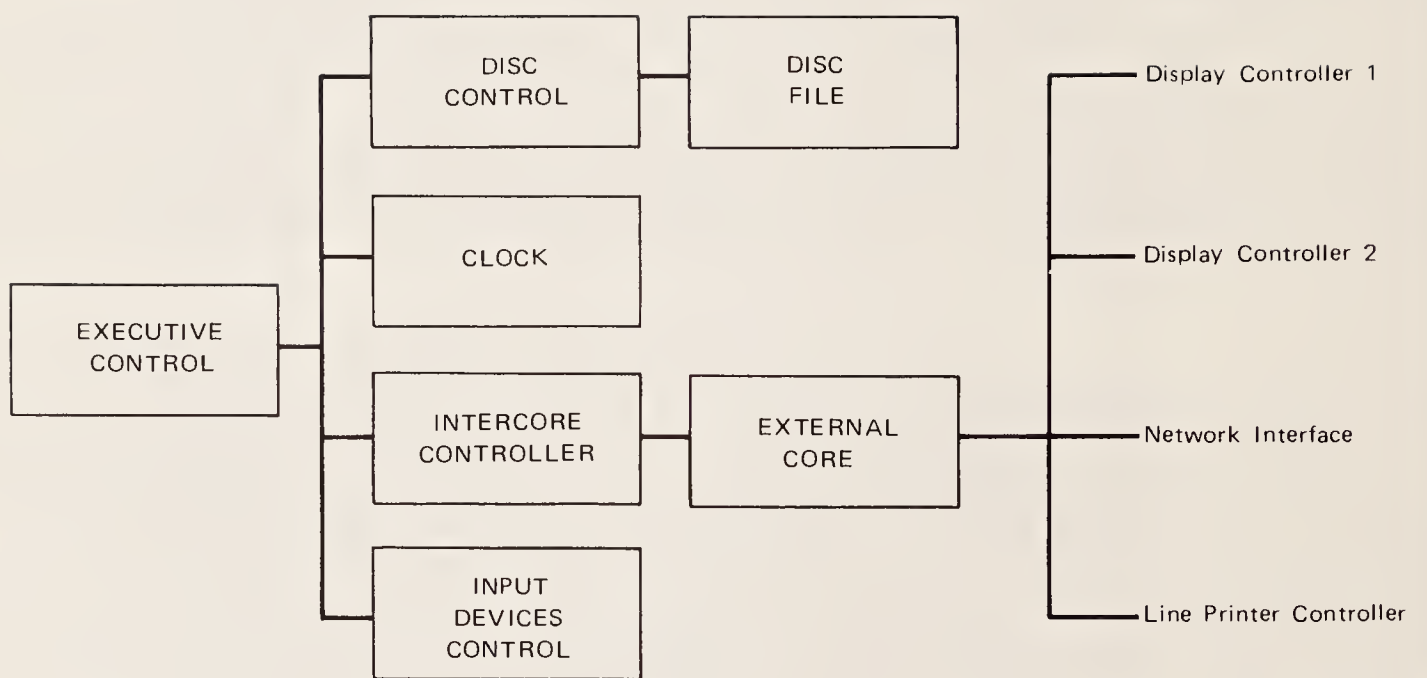
- (2) A short transfer mode allows the transfer of short, fixed-length buffers between fixed locations in 940 core and external core. This mode is easier to set up than the block transfer, and requires fewer memory accesses for control. It will be used for such functions as transferring single characters or other control information between the two core systems.

The operation of the inter-core controller is described in more detail in Appendix C.

The external core itself currently consists of a single 32,000-word bank with access switching to allow access by up to eight devices. Provisions are included in the design for expansion to 16 devices and two core banks of 64,000 words each. The core cycle time is 1.5 microseconds and the word length is 24 bits.

The interface to external core has been designed so that it is identical to the interface to 940 core (through the Executive Control). A device may be simply plugged into either core system.

As shown in Fig. III-3, we will initially be operating both display systems, the network interface, and the line printer



TA-7101-4

FIGURE III-3 SPECIAL DEVICES CHANNEL WITH EXTERNAL CORE

from external core. These are the devices that need constant buffers for relatively long periods and therefore require frozen pages when operating from 940 core.

2. Faster Drums

From the system response studies (see Sec. IV) it is apparent that a primary factor in response is the swapping bandwidth. To improve response (and add more users), we are in the process of replacing the XDS drums with Univac FH-432 drums.

These drums rotate at 7200 RPM, giving a transfer rate of 365,000 words per second (as compared to 120,000 for the present drums) and an average access time of about 4 milliseconds.

In addition, we are formatting the new drums in a way that will allow a page transfer to begin at any position on the drum. Since a 2048-word page fills two-thirds of a band, this will give an average page transfer time of about 8 milliseconds.

The interface for the drums will be designed and built by ARC. It will connect to the 940 through a second Memory Interface Connection (MIC), replacing the current RAD-DACC combination shown in Fig. III-1.

3. Clock System

An accurate clock system is being added to assist us in system measurements.

This clock system provides two types of time information - absolute and relative -- that are written into fixed locations in 940 core at regular intervals.

Absolute time consists of binary representations of year, month, day, hour, minute, and second.

Relative time information consists of a single 24-bit number, incremented and written into core every 100 microseconds.

The long-term drift on the clock will be less than 1 second in 250 days.

A more complete description of the clock system is given in Appendix C.

D. Notes on System Design and Reliability

1. Display System

The display system in use is somewhat unusual in that it uses central display-generating equipment and a closed-circuit television system to distribute images to the working area. This approach to a display system was chosen on the basis of cost and flexibility. A description of the system and of considerations that went into its design is given in an earlier report (Ref. 3).

We now have considerable experience in operating this system and are still very pleased with the basic approach, but we have had some problems with the component equipment involved.

The closed-circuit television system offers several distinct advantages over other means of producing displays at a work station.

The system is extremely flexible as to the location and design of working consoles, since only a television monitor and a video line are required to present the display at each console. This allows freedom to experiment with different types of consoles (Ref. 4) and to move consoles about without cabling problems.

The video signal is inverted to provide a black-on-white display. This presentation is usable in higher ambient light conditions than the usual bright-on-dark presentation, and flicker in the display image (due to low generation rates) is much less noticeable to the user.

With proper adjustment of the television camera, a significant storage time can be obtained on the vidicon surface. This greatly reduces the flicker effect that is present in the original CRT presentation. With this system we find it possible to regenerate displays at about 20 cycles per second.

Maintenance features are another significant advantage.

The display equipment at the actual work station is quite simple, consisting of only a television monitor which can be replaced by a spare for maintenance.

The display-generating equipment, which requires more

complex maintenance and repairs, is located centrally in the computer room. This makes it very easy to maintain an uncluttered office environment in the working area.

Furthermore, since there is not a fixed one-to-one relationship between display-generating equipment and work stations, when a portion of the display system is down for repairs the working consoles that remain operative may be freely selected on the basis of current needs.

Having two identical display systems, from display controller through actual monitors, has been a major factor in maintaining up-time in spite of the unexpectedly high level of maintenance required on the system.

The use of video to distribute display images offers several other possibilities that we have not yet fully exploited.

For the television monitor on which the image is presented, a wide range of accessory equipment is commercially available. For example, we have used high-quality projection television at the Fall Joint Computer Conference in 1968 and at the ASIS Conference in 1969. It is possible to use multiple TV monitors or intermediate-size projection equipment for smaller groups. This will be a major factor in the team-augmentation work to be carried out under the next contract.

The video capability offers additional flexibility in the images that may be used on the screen. For example, in the conferences mentioned above, live TV pictures of the people and equipment involved were freely used, mixed with the computer-generated image. This, again, will be a significant factor in team collaboration at a distance where pictures of the people involved can be used, either mixed or inserted with the computer-generated image.

Another use of the video that will become increasingly important is the viewing of microfiche documents. Many systems are now available and more are coming on the market for the storage, retrieval, and viewing of microfiche on closed-circuit television.

2. Maintenance Experience

a. General

In general the reliability of the facility has been very good; the computer up-time has been extremely high. The reliability of the disc-file system has been fair. We had period of several months of above-normal error rate, and 5 days down while clocks were rewritten; however, the trouble now seem to have been corrected.

One notable exception to this has been the line printer.

We originally bought a Potter chain printer which turned out to have marginal print quality and was ver unreliable. We had great difficulty in getting maintenance from Potter, and we finally replaced the unit with a Data Products drum printer. Like the Potter printer, this has 96 printing characters with upper- and lower-case alphabet. The print quality is excellent and so far it has been very reliable.

b. Display System

We have spent more effort on maintenance of the display system than any other part of the facility; since it is somewhat unusual, we will discuss some of the problems encountered and summarize the maintenance costs.

One of the basic limitations of the system is the lack o enough total light on the vidicon surface. This means that many design factors are marginal. The Tasker CRTs run at such high intensity that their life is relatively short. This high intensity also causes difficulties in maintaining good focus over the entire image. To operat with these low light levels, the vidicons must be quite sensitive; since sensitivity drops off with age, they have a relatively short useful life.

Because the writing speed of the Tasker display generators is lower than expected, we still have a flicker problem when all 6 screens on the system in use are reasonably full of text. To some extent we are able to compensate for this by careful adjustment of the vidicon beam current and target, but this adjustment needs frequent attention. We have considered longer-persistence phosphors on the TV monitors and will experiment with this in the near future.

In addition to these difficulties there are some basic weaknesses in the design of the Tasker system and the television system.

(1) Tasker System

Sockets for circuit cards are not of high quality. This results in contact-resistance problems, especially in the analog circuitry.

Deflection circuitry, with its many adjustments, is so hard to get at that it is left in a partially assembled state.

Logic circuits still do not have all pull-up problems corrected, resulting in a narrow range on the clock.

The active deflection-sensing circuit requires frequent adjustment.

The focus vs. beam position circuits perform very poorly.

(2) Television System

The preamplifier tubes on the television cameras tend to be very noisy. These tubes must initially be selected for low noise to get really good pictures, and their life is very short.

We are currently in the process of replacing all of the preamplifier circuit boards with a new solid-state circuit now delivered in new GE cameras of this type. This circuit uses an FET preamplifier with very low noise and hopefully no problems in reliability.

Controller power supplies are poorly designed and require too frequent replacement of parts.

c. Maintenance Costs

The following is a summary of the costs for maintenance of the display and television systems for the past year. Both include the frequent "tuning" necessary to maintain good picture quality. These are the costs for maintaining 6 operating work stations, but some effort has been spent on the equipment not in regular use. We expect this to go up

Sec. III
HARDWARE SYSTEM

about 50 percent when 12 stations are in operation.

TV System		
Labor	25,665	
Vidicons	3,365	
Picture Tubes	895	
Preamplifier Tubes	1,200	
All other parts	1,040	
Total		32,165
Tasker System		
Labor	7,905	
CRT's	3,000	
Miscellaneous	200	
Total		11,105

Note: The Tasker system is maintained at a "keep-it-going-well-enough-so-people-can-work" level, and it lives with many weaknesses.

3. Hardware Design and Construction Techniques

a. Logic Design Aids

The wirelist generator program described in an earlier report (Ref. 3) is still being used. The input format, diagnostic aids, and general form of the program are essentially the same as in the past. In the past the wirelist output was used to produce documentation that aided a technician in hand wiring; now it produces a punched tape that in turn controls a semiautomatic wire-wrapping machine. This wire-wrapping service is obtained from a local supplier and results in more accurate wiring, lower wiring cost, and faster turnaround in going from logic equations to finished wiring.

Regarding accuracy, no misplaced wires have been found to date, although a very minor number of broken wires and wires shorted to pins have been observed.

The wiring itself costs about 23 cents per wire. Also, above the cost of running the basic wirelist generator program, there is an additional cost of 20 cents per wire for preparing the paper tape used to control the wire-wrapping machine.

Turnaround time for wire-wrapping is short, typically less than a week for a design containing 400 integrated circuits. Of course, this is subject to

considerable variation, depending on the work load of the company performing the wire-wrapping.

Most of the general comments in the previous report concerning the utility of the wirelist generator program still hold.

However, experience has shown the desirability of maintaining a fairly complete set of logical schematics, complete with circuit locations and pin numbers, in addition to the designer's sketches and listings provided by the wirelist generator.

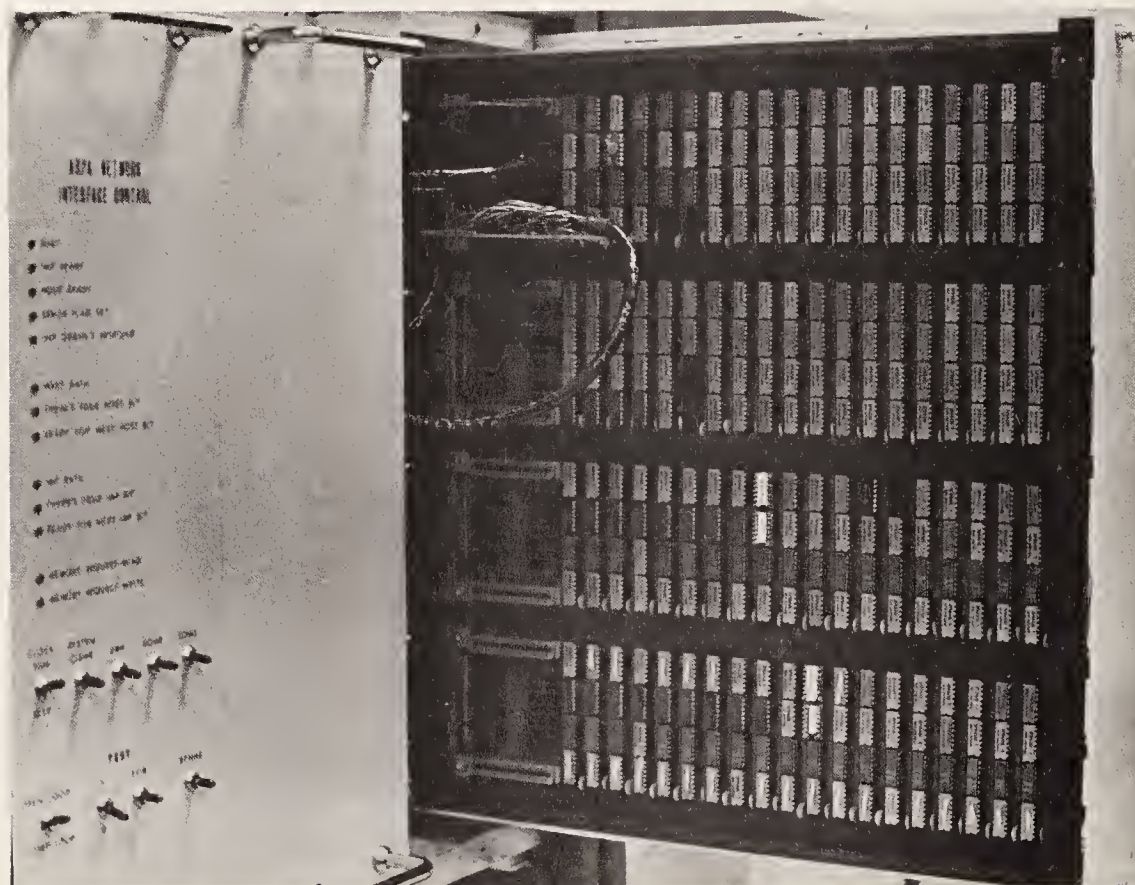
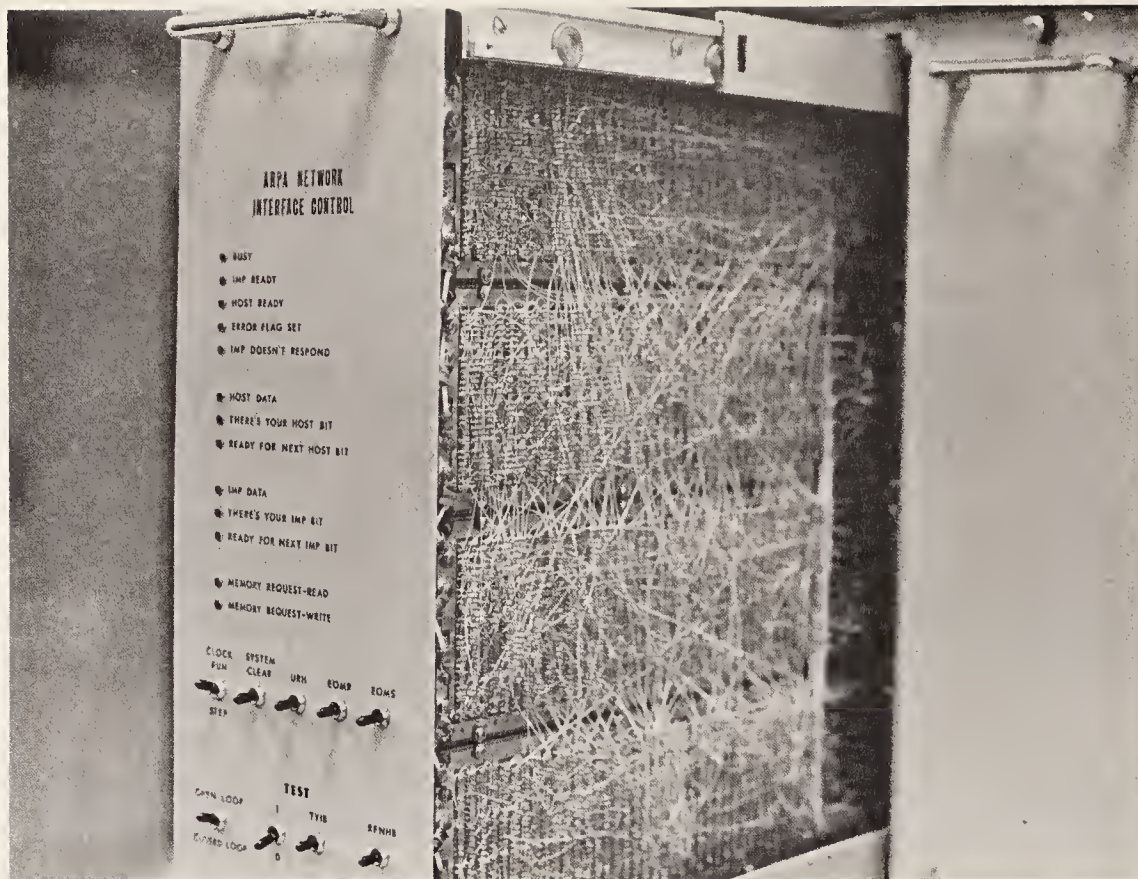
The previous report on this contract (Ref. 3) implied that the sketches and listing were sufficient for equipment maintenance and trouble-shooting. This is true as long as the original designer performs the maintenance. With the inevitable turnover of personnel that takes place on a long-term project, someone other than the designer eventually becomes responsible for keeping a given device operating. Under this circumstance, a schematic is an invaluable aid.

b. Construction Techniques

The construction techniques of the most recent units can be seen in Fig. III-4. The hardware implementation consists of an array of sockets that will directly accept a dual inline packaged integrated circuit (commonly called a "DIP"). The arrays of DIPs are mounted perpendicular to the horizontal plane on the front of the rack in which they are mounted. The circuit arrays can be pulled out for access. Wiring connections are made directly to the pins of the sockets. This scheme has several advantages.

First, the cost is low. The previous construction technique used printed-circuit boards for mounting the integrated circuits. Thus the cost of mounting the circuits on the board and the cost of the board itself were incurred.

Second, there is greater flexibility in the location of a given circuit type. With the integrated circuits mounted on printed-circuit boards, a complete board consisting of up to 12 circuits would have to be used in cases where only 1 circuit was actually needed.



TA-7101-30

FIGURE III-4 NETWORK INTERFACE CONSTRUCTION, SHOWING MOUNTING SYSTEMS FOR CIRCUIT ARRAYS AND MULTIPLEX SWITCH

Thirdly, an individual DIP can be removed and replaced. This is a great aid in the maintenance of a device. A DIP with a suspect circuit can quickly be removed and replaced by one that is known to be good.

In addition to the techniques of hardware realization of the basic logic design, many other details of the hardware design are important.

One feature that the hardware must provide is some means of access to both the integrated circuits and the wiring -- this feature is an absolute necessity during initial checkout and is an aid in later maintenance and changes.

In providing access to the external core, the multiplex switch posed a particularly difficult problem, since 34 cables connect to it. In order to allow easy access to this unit, the mounting system shown in Fig. III-4 was developed.

A very flexible cable is used, with a rather elaborate method of strain relief and cable guidance. Although the original mechanical design was quite expensive, requiring about 3 months of a design draftsman's time, past experience has shown the difficulty of maintaining equipment that did not have easy access. To date this design cost has been spread over several units and its anticipated use in future units will reduce the per-unit cost for the design. The expense of hand-fabricating the parts for a pull-out drawer is estimated to be around \$300, which is slightly less than \$1 per socket.

In the recent equipment, light-emitting diodes (LEDs) have been used instead of incandescent lights for panel indicators. The results have been very satisfying.

The LEDs have a higher initial cost (about \$3 each) than the incandescent lights previously used. The lights, however, have a limited life while the lifetime of the LEDs is essentially infinite. This leads to essentially zero maintenance and replacement cost for the LEDs.

This long service life also means that the expensive sockets required by the incandescent units, in order to facilitate their replacement, can be eliminated. Indicators were mounted simply by drilling holes in the front panel and retaining the LEDs with RTV silicone

Sec. III HARDWARE SYSTEM

rubber.

A further cost saving is effected since these lights are driven directly from the logic, saving not only the cost of the drivers themselves but also the cost of the extra sockets and wiring they would require.

The LEDs have a relatively narrow viewing angle and less intensity than the incandescent lights, but we have found them entirely satisfactory in use.

c. Typical Construction Costs

A fairly careful study was made of the actual cost of the ARPA Network interface. This is typical of the type of control unit that is now being built.

Hardware and Construction -- the figures are given on a per-socket basis. Technician time involved in construction is included.

Frame, connectors, IC sockets, etc.	\$3.50
Mounting hardware	\$2.00
Computer time (preparing wire-wrapping control tape, 35 cents per wire and an average of 6.8 wires per socket)	\$2.40
Integrated circuits (average)	\$2.00
Wire-wrapping (25 cents/wire and 6.8 wires/socket)	\$1.60
Total hardware and construction (per socket)	\$11.50
Total hardware and construction cost for Network interface (600 sockets)	\$6900.00

Design

The design cost is expressed in man-days for a design engineer.

Initial design	10 days
----------------	---------

Sec. III
HARDWARE SYSTEM

Preparation of equations	10 days
Drawings and documentation	10 days
Final assembly and debug	20 days
Total	50 days

A. Introduction

The central focus of software activity at the Augmentation Research Center is the evolutionary development of the On-Line System (NLS), and during the contract period this work has continued in the spirit of bootstrapping which has been consciously applied since the project's inception. In addition to RADC funding, this work has received substantial support from NASA under Contract NAS1-7897.

The original version of NLS (then called NLTS for On-Line Text System) resided first in a CDC160A computer (Refs. 5 and 6); it was later transferred to a CDC3100 on which further development took place (Ref. 7).

The experience and tools developed with the 160A and 3100 systems were then applied to the design and construction of the present NLS, which provides multi-console service from an XDS940 computer and associated special-purpose hardware.

As has been true throughout its development, the On-Line System is now being used principally as an instrument for planning and engineering its own evolution and as a tool for composing, editing, and publishing documents (such as this report) for distribution outside of the Center.

The operation and evolution of NLS takes place within a rich environment of software systems, many of which were created specifically to aid in its development.

Most basic to the operation of NLS is the timesharing system (TSS) running on the XDS940.

TSS was originally developed by Project GENIE at the Berkeley campus of the University of California, but responsibility for maintenance of the ARC version presently lies with the Center itself.

Each user runs NLS as a subsystem of TSS and consequently has access to other TSS subsystems such as the KDF file system, the QED text-handling system, and the DDT symbolic debugging system.

Work done on TSS during the contract period is described in Section IV-B.

Sec. IV SOFTWARE SYSTEM

The evolution of NLS has been facilitated greatly through the use of an extensive collection of languages and their respective compilers, most of which were developed by ARC itself. These languages and compilers are discussed in Section IV-C.

The program code for NLS resides in such a large number of files that compiling, loading, and debugging the system is a complex process. To make these operations more manageable, a TSS subsystem called NLS UTILITY (not to be confused with the internal utility routines of NLS itself) has been constructed during the past year. A description of NLS UTILITY will be found in Section IV-G.

During the contract period extensive changes have been made to NLS, both in user service features and in internal system organization.

Development was begun on the Typewriter-Oriented Documentation-Aid System (TODAS), which will make much of the power of NLS available to users at remote locations through hard-copy terminals such as Teletypes. Implementation of TODAS is one of the major steps being taken in setting up the Network Information Center (NIC) for the ARPA Network.

The ability to examine the contents of NLS files has been enhanced by the implementation of a powerful set of JUMP commands, including provision for jumping between files using file links. (A file link is simply an occurrence of a file name, properly embedded within the text of another file.)

Facilities have been provided to enable the NLS user to request that each file statement displayed be tagged with the initials of the person who last modified that statement along with the date of modification.

Conventions for handling keyset input have been changed so that the 31 input characters may be interpreted in any of four cases (lower case, upper case, numbers and special characters, and VIEWSPECs). The case is determined by concurrent input from the center and left pushbuttons on the mouse (lower case is the normal case).

Commands have been added to enable the user to set any text entity in a variety of type styles (upper case, lower case, italic, boldface, flickering, underlined), and the display-generation routines have been modified so as to display text in the specified forms.

A limited output-processor capability has been provided so that programs maintained as NLS text files can be compiled directly from NLS (rather than having to be converted to QED files first).

Several other new features have been added to NLS, including the following:

- (1) Vector package -- a basic graphics capability permitting the user to insert simple line drawings into a file
- (2) Keyword system -- a means of information retrieval working upon special information inserted in a file, with user control over categories of information to be retrieved
- (3) Calculator package -- a calculation capability for the NLS user, providing four storage registers and an accumulator, ADD, SUBTRACT, MULTIPLY, and DIVIDE operations, and the ability to select operand numbers from file text and insert results back into the file text
- (4) Substitute command -- causes automatic substitution of one user-specified character string for another, throughout some user-specified portion of the file
- (5) File cleanup and compaction -- automatic user-controlled correction of certain kinds of system-caused errors in a file, and reduction of the storage needed for the file by means of special garbage-collection methods
- (6) Output of NLS files to microfilm (via an out-of-house facility).

In addition, the overlay structure of NLS has been reorganized to provide room for growth of the system, and numerous other internal system changes have been made to provide improved service and reliability.

An overview of the current structure of NLS is provided in Section IV-E, and a more detailed description will be found in Appendix D.

Descriptions of earlier work on the design and development of NLS for the XDS940 are contained in Refs. 7, 8, and 9.

Other software development activities covered in this report include preparations for interfacing with the ARPA Network (see

Sec. IV SOFTWARE SYSTEM

Section IV-F), and a simulation study of factors affecting the response time of the timesharing system when a number of NLS users are being served (see Section IV-D).

B. The Timesharing System (TSS)

The support of new hardware and improved response to the NLS users are the two main reasons for the expenditure of effort on the timesharing system (TSS).

1. Disc Support

The Bryant disc device was received in August 1968. This device has the capability of storing 32 million 24-bit words. With the acceptance of this device, a file-storage program called KDF was implemented to provide users with a means of storing information. The earliest form of KDF operated essentially independently of the TSS I/O handling system. A later version was integrated with the TSS system, and made all accesses to the disc via calls on the supervisor.

During late 1968 and the early months of 1969, the TSS system was extensively modified to include scratch disc files. These files are handled by the same calls on the supervisor as are the drum files. In this way, the disc files have the flexibility of the drum files as well as freeing the user from KDF's restrictions on the number and size of files. Disc scratch files may be used for all the same functions as drum files, while KDF is used primarily for storage. The disc file space is pooled by all the users and thus has the additional advantage of more economical use of this space than is possible under KDF. The development of improved garbage-collection facilities permitted the use of "permanent" scratch files on the disc for longer-term storage of heavily used files.

2. Magnetic Tape Support

The new TSS developed in late 1968 and early 1969 incorporated the direct tape I/O package, which permitted more efficient use of tape files. The increased speed and efficiency of the tape files made it more practical to copy information stored under KDF to magnetic tape, thus protecting this information from loss in the event of serious disc failure.

Further work has been done to improve the reliability and speed of access of tape files, as required by the Archive/Journal system (see Appendix B). The magnetic tapes serve as the main storage facility for most of the older or less used files, and

thus relieve KDF of the burden of storing these files.

3. External Core

The inter-core controller (ICC) and the external core memory became available in early 1970. Several supervisor calls have been written to allow the user to access this device.

TSS allows a user to obtain up to 16 thousand words of external core memory, and maintains tables which perform a limited relabeling function between user-provided addresses and physical addresses.

Other calls permit the user to make data transfers via ICC between external core and 940 memory and vice versa, as well as transfers from one area of external core memory to another area of external core memory, or from one area of 940 memory to another area of 940 memory.

4. Other Devices

A program has been written to permit the queueing of print files. This program allows the user to place his file in a print queue and continue on to other tasks. The queueing program informs the user of his file's position in the printer queue and the approximate amount of material to be output before his file will be completed.

Minor additions and modifications to the TSS system have been made to support the Data Products printer and several new Teletype and typewriter-style terminals.

5. Research on Scheduling Algorithms

The system simulation (discussed in Sec. IV-D) has indicated that system response to the NLS user might be improved by redesign of the scheduling algorithm. Toward this end, we have experimented with several modifications to the scheduling algorithm, particularly with respect to the assignment of priorities and the queue-assignment schemes.

One such experiment consisted of assigning a special queue for NLS users, giving them higher priority than other I/O users or users who place heavy computational loads on the system.

This queue measurably improved the response for the NLS user, but so impaired the response to other users that in some cases it was not possible to run the executive

programs.

Since that early trial, we have implemented a new scheme that favors NLS users and any other users who are engaged in frequent but short I/O processes. The improvement has not been as noticeable as with the earlier scheme, but has not resulted in such severe impairment of service to other users. This algorithm tends to favor the user who is engaged in editing text, as opposed to the user who is doing a great deal of file manipulation. Another part of this effort has shown that another queue was not serving a useful purpose, and this queue has since been discarded.

6. General

Much work has been done in restructuring the TSS system to provide space for accommodating the storage requirements of the ARPA Network. Several routines have been rewritten and moved to the Executive, and others have been moved to nonresident pages. In this way, several hundred core locations have been made available for Network use.

Because of the greatly reduced level of effort of Project GENIE at UC Berkeley, it has become necessary for us to further the development of TSS essentially independently.

C. Compilers

1. Introduction

The development of NLS has been greatly facilitated through the use of a powerful complement of languages and compilers, most of which were designed at ARC.

The languages used range in generality from the NARP assembly language through a collection of special-purpose languages (SPL's) unique to NLS implementation.

Having such a flexible set of languages from which to choose makes it possible to select for each programming task the language in which the desired operations can be expressed most naturally.

a. NARP

There are a few parts of NLS that can be most conveniently coded in assembly language (e.g., the data page and the display-buffer page), and for these the NARP assembly

language is used.

Also, for historical reasons, the timesharing system (TSS) and most of its subsystems (e.g., KDF and DDT) are coded in NARP.

The NARP assembler is based on another assembler, AKPAS; both of these languages were produced by Project GENIE for use in the development of TSS (see Refs. 10 and 11).

b. MOL940

MOL940 (or simply MOL) is a machine-oriented language for the XDS940 and was created by ARC to aid in the programming of NLS.

MOL combines the flexibility of assembly language with the algorithmic clarity of higher-level procedure-oriented languages. Much of NLS is coded in MOL.

The original version of MOL940 is described in Ref. 12, while this report contains a brief description of the current version.

During the contract period MOL has been substantially rewritten to improve its performance and provide new programming features.

The current MOL compiler was produced using the new version of Tree Meta (described below); consequently, the MOL compiler now generates binary machine code directly rather than producing assembly-language code.

As a result of this change, assembly-language instructions are now treated as built-in functions, whereas previously they were handled using escape conventions which provided for them to be passed directly into the output stream without translation.

Optional mechanisms have been added to facilitate the writing of reentrant code, using a software stack for procedure calls and for storage of local temporaries.

The syntax for procedure calls has been modified so that an entire NLS file link may be used in place of the procedure name alone.

The presence of the file link augments a programmer's

ability to study a complex system of programs occupying several NLS files, by making it very easy for him to jump from a file containing a reference to some procedure into the file containing the procedure itself. In compiling a program only the name part of the file link is used; the rest of the link is treated as commentary information, since it is irrelevant to the compilation process.

Tree Meta

Tree Meta is a compiler-compiler developed at ARC; it is used to produce compilers for MOL and all the special-purpose languages (and for itself as well).

Section IV-C-2 contains a brief overview of the current version of Tree Meta, and a more detailed description is in preparation for release as a separate report. (Pending publication of the Tree Meta document, a description more complete than that contained in the present report can be found in Ref. 8.)

During the contract period, the only major change to the Tree Meta system was a modification to the basic way in which compilers produced by Tree Meta generate code.

Compilers produced by Tree Meta used to translate a given source language into assembly language, which then had to be translated by the NARP assembler to obtain machine code.

With the new Tree Meta, the compilers generate machine code directly, thus eliminating one step of the translation process.

The SPL's

Many of the higher-level operations of NLS are carried out by programs written in one of a set of special-purpose languages (SPL's). Each of these languages is translated into machine code by a compiler produced with the Tree Meta system.

Each SPL represents an attempt to formalize a particular function of NLS, aiming at a syntax appropriate to the data base and operations required for NLS, while at the same time embodying the potential and peculiarities of the XDS940 computer.

The four SPL's currently in use are the input-feedback language, the structure-manipulation language, the content-analysis language, and the string-construction language.

Detailed descriptions of the SPL's will be found in Appendix D of this report as well as in Ref. 8.

Although extensive changes in the SPL's are planned for the near future, no basic conceptual changes were made during the contract period.

2. Tree Meta: A Compiler-Writing System

A compiler-writing system was implemented within the ARC for use in writing compilers for the MOL940 language and the special-purpose languages (SPLs) used in implementing NLS.

The Tree Meta language allows one to concisely specify the syntax of a language, in a notation similar to Backus-Naur Form. Embedded within this syntax specification are rules and directives describing exactly how the compilation of a program written in the language is to take place.

The Tree Meta compiler reads a textual program written in the Tree Meta language, and directly produces a binary machine-language program which is a compiler for the specified language. The new compiler is then capable of reading a textual program in the specified language and producing a binary program according to the compilation rules embodied in the compiler.

Tree Meta is expressed in its own language, and is thus self-compiling. The current version has been produced from previous, more limited versions by the process of bootstrapping.

Tree Meta has proven to be a particularly valuable tool in system development at ARC, because of the experimental nature of the development being done here.

Perhaps the most valuable feature of Tree Meta is its ease of use. A complete compiler description is contained in a single text file and is readily edited and recompiled. A change in a compiler can be tried in two or three minutes. This allows experimentation that otherwise would be too time-consuming, and makes the debugging of language specifications quite fast. This flexibility is very

important when a language is being developed -- as opposed to having been prespecified and fixed in its definition.

The relatively simple Tree Meta notation describes a language precisely, and anyone familiar with the notation can see what the syntax is. The code for the compiler is also the formal definition of the language to be compiled.

Also, since the source code for the Tree Meta compiler is simply a description of the Tree Meta compiler expressed in the Tree Meta language itself, it is possible to produce a new version of Tree Meta merely by editing and recompiling this description.

The Tree Meta system consists of this symbolic description, the Tree Meta compiler, and a library of support routines in MOL. The support routines perform functions such as input/output and symbol-storage operations.

The Tree Meta compiler is relatively fast. It compiles itself in about 30 seconds from about 8 pages of text input. The compiled program is about 12 thousand words of memory, including tables and storage areas.

In the formalism of Tree Meta, a compiler consists of (1) parse rules, which parse the input in a top-down manner and build a tree structure, and (2) unparse rules, which then test the tree structure and produce machine code. The tree consists of symbols taken from the input, values and flags inserted in the tree by the parse rules, and nonterminal nodes that correspond to unparse rules.

The parse rules test the input stream to identify the constructs it contains.

For example, to test the input stream for an assignment statement, the following rule called "assign" might be used.

```
assign = identifier "<" expression :store/2/;
```

This parse rule defines an "assign" to be an "identifier" followed by a left-arrow followed by an "expression," where "identifier" and "expression" would be defined by other parse rules.

If the input stream is matched by this rule, a node will be constructed in the tree and tagged with the name

"store."

This node will have two nodes under it, corresponding to "identifier" and "expression," respectively.

The unparse rules are executed beginning with the last node built into the tree. The node names in the tree determine which rules will be invoked to compile code from that node of the tree.

In the example above, the unparse rule named "store" will test the node for several different forms and output code depending on the form. A test might be

```
[identifier,add(*1,-)]
```

This test reads as follows: The "store" node must have two nodes under it. The first node must be an identifier. The second must be a node named "add," which has two nodes under it. Furthermore, the first node of "add" must be exactly the same as the first node of "store." This test would be satisfied by input of the form

x + x + (anything)

Another test might be

```
[identifier,add(*1,"1")]
```

which is the same but with the additional requirement that the second node of "add" must be the number "1". This is checking for input of the form

y + y + 1

The unparse rule "store" might begin:

```
store    [identifier,add(*1,"1")] => MIN *1,    ;
```

```
        [identifier,add(*1,-)] => lda(*2:2) ADM *1,    ;
```

If the test on the first line succeeds, "store" produces a single memory-increment instruction, MIN, operating on the memory word addressed by the identifier (the first node of "store"). Otherwise, if the second test succeeds, an unparse rule named "lda" is called with the second node of "add," as argument in order to produce code to load the A-register. Then an add-to-memory instruction is produced,

again operating on the memory word addressed by the identifier. The rule "store" would then continue by testing for other forms of expressions, until all legal forms have been taken care of.

The tree serves as an intermediate form of the program -- form which facilitates extensive testing by the unparse rules, and which usually contains no redundant information. The compiler author determines the forms of the trees completely when writing the compiler. His ingenuity in determining the tree forms and compilation schemes is generally not restricted by the Tree Meta language.

Symbols (which may be of arbitrary length) are read from the input and kept in a symbol-storage area where they are referenced via a hash table. Symbols may also be created and entered into the symbol-storage area by the compiler. Each symbol has a 24-bit value as well as 24 attribute bits. The meanings for most of the attribute bits may be defined by the compiler writer, and symbol values and attributes may be set, reset, and tested during the running of the compiler.

The output from any Tree Meta generated compiler is a relocatable binary file, produced in the proper form for DLI (the loader and debugging system). This binary file includes the symbols from the program, so that programs can be debugged symbolically.

3. A Machine-Oriented Language, MOL940

In spite of the quite sophisticated understanding of compilers and compiler-compilers in computer science, assembly language is still used for the bulk of system programming.

ARC has used a machine-oriented language as a replacement for assembly language in the writing of system programs. The machine-oriented language, MOL940 (or simply "MOL") offers the power of an assembly language while providing the algorithmic clarity found only in a higher-level language.

A machine-oriented language is designed to give the programmer a block-structured language with many of the usual associated features, such as conditional and iterative statements, subscripting, and arithmetic expressions.

At the same time, the language is designed to reflect the idiosyncrasies of the actual machine on which the programme

is writing his programs. To this end, special constructs are incorporated in the language which allow the programmer to have some control over the code which is produced and the manner in which the central registers are used.

The idea of a machine-oriented language is not new.

Erwin Book of System Development Corporation first developed an MOL for the Q-32 and later an MOL for the IBM 360.

Niklaus Wirth's PL-360 was an MOL used to implement a version of ALGOL on the 360.

An MOL for the XDS940 was a early development of ARC, and was used in the initial implementation of NLS. A modified version of this language, developed with Tree Meta, is the MOL described in this section.

The general design of MOL940 is actually machine-independent. Only the inclusion of special logical forms and built-in functions gives the language a specific orientation towards a particular machine. Thus it may serve as a basis from which MOLs for other machines may be derived by substituting other logical forms and other built-in functions.

Among the distinguishing factors of any programming language are the means provided for referencing information and for controlling the flow of execution.

In MOL940 the means for referencing information are as complete as in an assembly language.

The central registers of the machine are represented as basic elements in the syntax of the language. Thus ".AR" stands for the A-register, ".AR+1" causes a 1 to be loaded into the A-register, and "X+.AR" causes the contents of the A-register to be stored in location X.

Assignment is made one of the binary operations that can occur in an arithmetic expression.

This allows the programmer to refer to the value of subexpressions in a very straightforward manner.

For example, one can write "k+(j+n)+10; or "k+10+j+n;" instead of "j+n; k+.AR + 10;". While both forms would result in the same code, the use of assignment as a binary operator avoids the explicit reference to the

A-register.

An apostrophe followed by a single character may be used interchangeably with the numerical code for that character.

This can be of great value in clarifying the intent of a test. For example, assume that the numerical code for a question mark is 16. Then a test for a question mark may be made by "'?' rather than the less informative "=16".

The term "literal" will be used to denote a term that can be either a number or an apostrophe followed by a single character.

Two modes of referencing information are provided to give addressing completeness. These modes are similar to the "left-hand value" and "right-hand value" concepts found in CPL and BCPL.

The modes are differentiated by the presence or absence of dollar sign in front of the reference. The former will be called "dollar mode." and the latter "normal mode." The values referenced by identifiers, literals, and strings in the two modes are as follows:

(1) Normal Mode

- (a) Identifier: contents of the cell whose address is the value of the identifier.
- (b) Literal: the numerical value of the literal
- (c) String: contents of the first cell used to hold the string

(2) Dollar Mode

- (a) Identifier: the value of the identifier (i.e., the address of a memory cell)
- (b) Literal: contents of the cell whose address equals the value of the literal
- (c) String: the address of the first cell used to hold the string.

The term "value of an identifier" as used here is equivalent to the left-hand value of an identifier in CPL.

Thus if cell 400 corresponds to the identifier k or if k has been set equal to 400, as in an EQU statement of an assembler, then the value of k is 400. It might also be called the symbol-table value of the identifier.

Notice that the normal mode of an identifier or literal corresponds to usage in problem-oriented languages.

Indexing and indirection are allowed where appropriate with the above forms.

Indexing is specified by following the reference with an expression enclosed in square brackets, while indirection is specified by enclosing the entire reference in square brackets.

The syntax disallows such dubious constructs as indexing with a literal or indirection with a string. The following shows in which cases indexing and/or indirection are allowed.

(1) Normal mode

- (a) Identifier: indexing and indirection
- (b) Literal: neither
- (c) String: indexing

(2) Dollar mode

- (a) Identifier: neither
- (b) Literal: indexing and indirection
- (c) String: neither.

The means mentioned above make an MOL at least as powerful as an assembly language in referencing information. In specifying the control of activation flow, an MOL is clearly superior.

Flow of activation is determined by the results of logical tests. It is in the clarity of expression of these logical tests that an MOL is particularly valuable.

To facilitate congruence between program construction and the idiosyncrasies of a given machine, the syntax of an MOL should contain constructs that reflect the logical tests

made possible by the instruction set.

For example, the XDS940 has an instruction that skips if the contents of the A-register and the effective address do not have ones in any corresponding bit positions. Thus MOL940 has a logical construct "Sum1 CB Sum2" which is true if and only if Sum1 has a one in a common bit position with Sum2.

In addition to logical constructs, there must be means to specify the repeated execution of a given statement and the choice for execution of a particular statement out of several. The main constructs for repetition in MOL940 are the LOOP and WHILE statements.

The LOOP statement is based on a suggestion of Knuth. It provides the most general possible form of control of repetition.

The statement following the word "LOOP" is executed repeatedly until an "EXIT" statement embedded within the loop is executed.

Execution of an EXIT statement causes control to leave the innermost LOOP containing it.

There may be an arbitrary number of EXIT statements within a LOOP, placed arbitrarily, and nested within blocks to an arbitrary level.

The WHILE statement simply serves as a convenient alternative way of writing a commonly used form of the LOOP statement, namely the form with a single EXIT occurring at the start of the LOOP.

Selective execution is provided by IF and CASE statements.

The IF statement is the standard Algol-like IF with an optional ELSE part.

Since the 940 uses skip instructions for logical tests, it is possible to optimize the branches required if there is no false part and the true part consists of a single instruction. This is done if the user writes "DO-SINGLE" instead of "THEN".

The CASE statement corresponds to a special form of the IF statement in which the case is selected for execution

according to the class into which an expression falls.
The syntax is roughly

"CASE" expression "OF" sequence of cases "ENDCASE"
statement

where each case in the sequence consists of one or more
tests followed by a statement.

A test consists of a binary-relation symbol followed by
the right-hand side of the binary relation. The test is
true if the binary relation formed by using the
expression at the head of the case as the left-hand side
is satisfied.

The first case with a true test is the one executed. If
none of the tests are true, then the statement following
"ENDCASE" is executed.

A common use of the CASE statement is in determining the
proper response to a character input from a terminal.

Finally, MOL940 permits the use of machine instructions as
built-in functions. The syntax of such a built-in is
roughly

function-name address-reference actual-arguments.

The function name is simply the standard mnemonic operation
code for the instruction.

The address reference is optional; if present, it may be an
identifier, literal, or string, with optional indexing or
indirection.

The actual arguments are also optional; if present, they
consist of a sequence of expressions to be loaded into
registers, separated by commas and enclosed in parentheses.

Such a built-in function may be used either as a statement
by itself or as a primary in an arithmetic expression.

It should be clear that this allows the programmer complete
access to the instruction set of the machine and gives the
opportunity to produce as efficient code as could be done in
assembly language (where this is deemed necessary).

Experience at ARC has shown that machine-oriented languages are

an attractive medium for systems programming. They permit efficient code, unrestricted data structures, and complete use of the machine instruction set, giving a flexibility usually associated only with assembly languages, while still providing the algorithmic clarity of higher-level languages.

D. Response Studies

We conducted a study of factors affecting the response time of the timesharing system on our XDS940 computer, which serves a number of NLS display terminals requiring very rapid response to user actions. The method of approach was a highly parameterized simulation of the timesharing system, which permits experimental evaluation of various possible methods of improving system response time. A summary of the approach and the results is given here.

1. Objectives of the Study

Although this study was conducted specifically on the timesharing system in use at ARC, it is of general interest (1) because of the unique method of approach, which permits easy implementation of results, and (2) because it may be expected that systems resembling NLS in some ways will be coming into more general use in the future. The principal characteristic of NLS that affects the behavior of the timesharing system is its dependence on fast, highly interactive operation of display terminals, and computer technology is already responding to a strong demand for this kind of user interface.

It should be emphasized that we are dealing here with the time required for the system to respond to individual commands from interactive users, and not with the system's speed in performing large numerical-computation tasks.

Interactive display usage for text manipulation, if it is to be really effective from the user's point of view, requires much shorter response times than have normally been considered satisfactory for timesharing systems; in the case of NLS, the desired response time for a typical command is a fraction of a second -- delays of more than a second can seriously impair the user's task performance if they occur too frequently. By contrast, the response of a less interactive system such as TODAS, which is not designed around an interactive display, is considered satisfactory if the typical delay in executing a simple command is no more than a few seconds.

The immediate goal of the current study is to develop an

understanding of the interrelated factors affecting the response time of AKC's timesharing system and to identify possibilities for modifying the hardware and software of the system so as to improve the responsiveness of this system.

2. Approach

The approach taken was to write a simulation of the timesharing system (TSS) operating on the XDS940. The simulation incorporates the scheduling and swapping algorithms of TSS and allows changing of parameters to represent various facility configurations and usages.

This allows an evaluation of the impact of changes in the hardware configuration, such as faster drums or larger core memory, as well as the effect of various mixes of user demands on the response of the system.

In addition, the program was written in such a way that with minor modifications, the simulation of the scheduler and swapper could become part of an actual timesharing system monitor. Thus changes in the scheduling and swapping algorithms can be tested by simulation and, if they prove to be valuable, incorporated into the actual system.

3. Results

Throughout this section the number of users is assumed to be equally divided between TODAS and NLS unless otherwise stated. In giving the results of the study, the average and the 80-percent delay times are used rather than the maximum.

a. Standard Parameter Values Used for Simulation

Hardware Parameters

Memory size: 32 pages, less 7 pages for resident monitor and less 1 page for each NLS user (for display buffers)

Drum latency: 17 msec

Transfer rate: 17 msec

File reference time: 30 msec

CPU speed: XDS940.

Software Parameters

Short quantum: 1/4 second

Full long quantum: 1 second.

User Parameters

3 user types: NLS, TODAS, and OTHER

64 tasks for NLS

32 tasks for TODAS

1 task for OTHER

The task descriptions for NLS and TODAS are based on studies of the actual systems.

b. User Types Considered in Simulation

In the actual use of the simulation, three types of users were considered.

Two of the types correspond to users of the two subsystems NLS and TODAS.

Users of type NLS or TODAS are assumed to be working steadily and at a relatively rapid pace, but their work is also assumed to be limited to tasks that do not require large amounts of computation to complete.

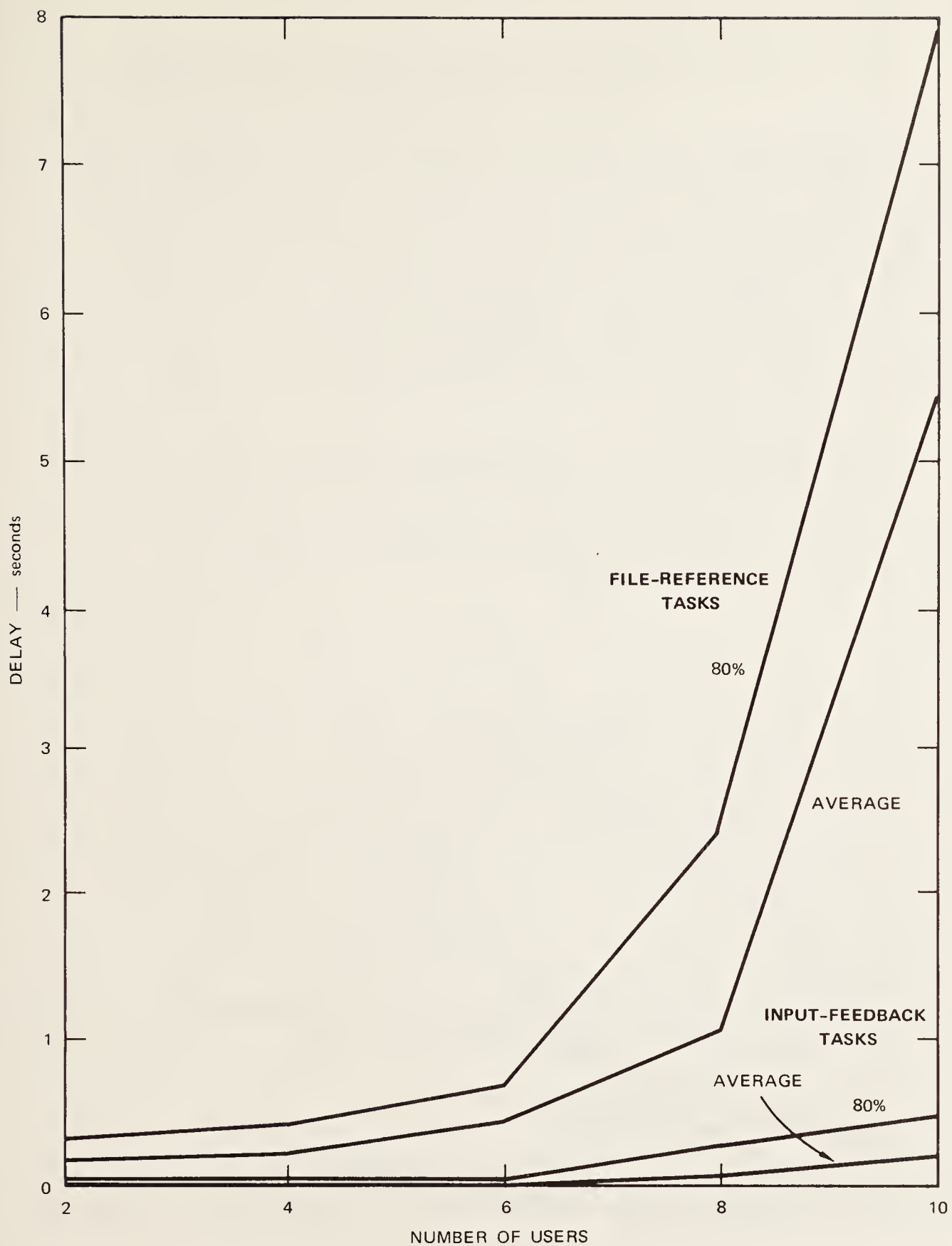
The third type of user is called OTHER, and is assumed to be working on tasks that consist of large amounts of computation. Compilation is an example of this kind of task.

One of the main concerns that prompted this study was to find means to maintain fast response for users of type NLS, and to a lesser degree those of type TODAS, when users of type OTHER are on the system.

c. Simulation of Current System

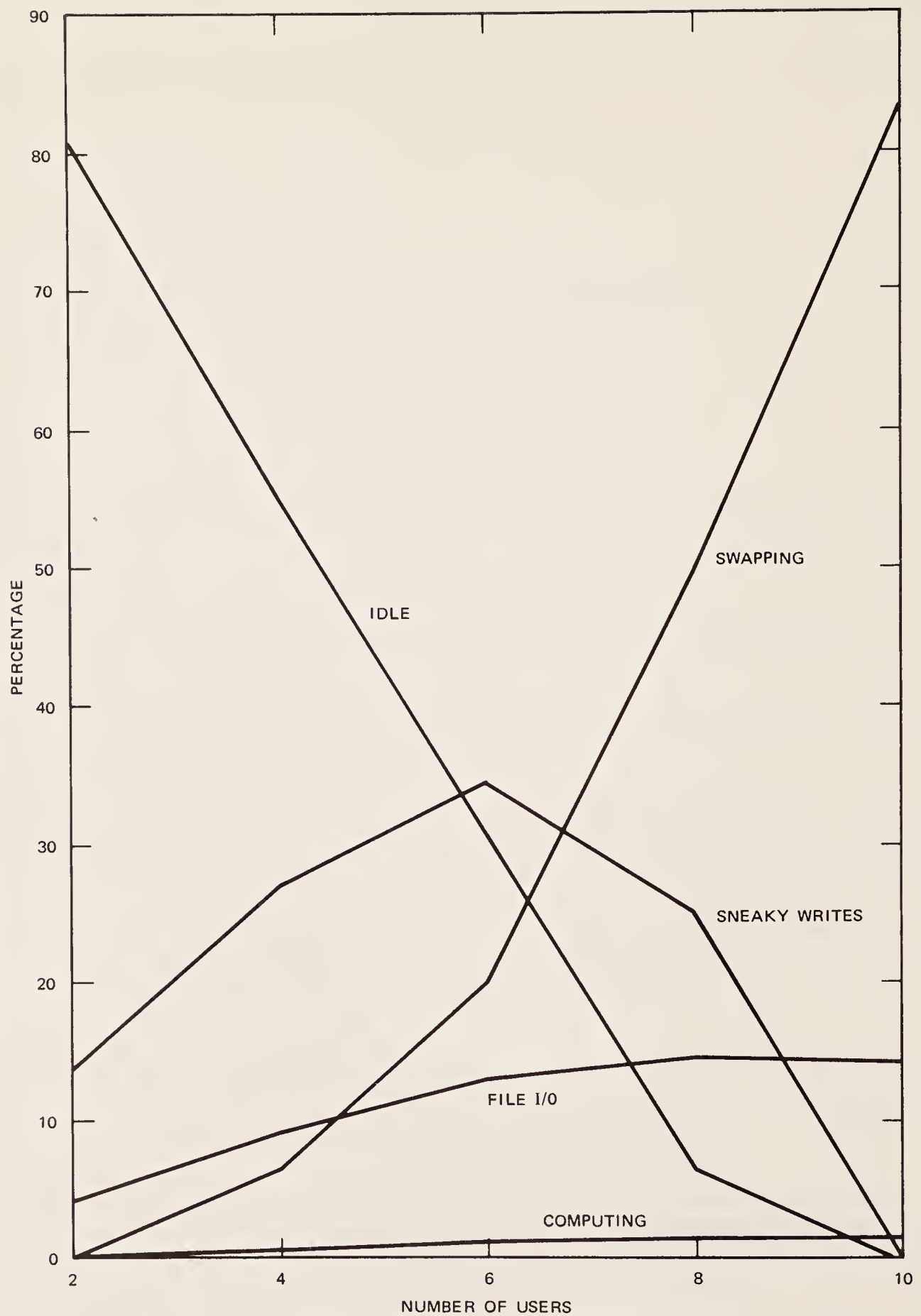
The facility assumed in this simulation has 64K of core memory and swapping drums with 4.5-megabyte total capacity.

Two views of the results of this simulation are shown in



TA-7101-6

FIGURE IV-1 CURRENT SYSTEM: AVERAGE AND 80-PERCENT DELAYS FOR NLS INPUT-FEEDBACK AND FILE-REFERENCE TASKS
—USERS EQUALLY DIVIDED BETWEEN NLS AND TODAS



TA-7101-7

FIGURE IV-2 PERCENTAGE OF TIME SPENT IN VARIOUS SYSTEM FUNCTIONS—USERS EQUALLY DIVIDED BETWEEN NLS AND TODAS

Figs. IV-1 and IV-2. For both of these the number of users is assumed to be equally divided between types TODAS and NLS, with no users of type OTHER.

Figure IV-1 shows both the average and the 80-percent delays for NLS input-feedback and file-referencing tasks. In the current system, the data for file referencing indicate the kind of delay experienced by a user when he asks the system to perform an editing function or to display a different section of his text. These results are very consistent with actual experience on the system. In actual use, subjective evaluation leads us to conclude that the system becomes virtually unusable when the delays as shown in this figure exceed about 2 seconds.

Figure IV-2 shows how the time distribution varies as the number of users increases. It is interesting to note here how quickly the swapping delays become the major factor in affecting response time and how small the delays due to computation time are. Section IV-D-3-f below goes into more detail on the effect of computation time.

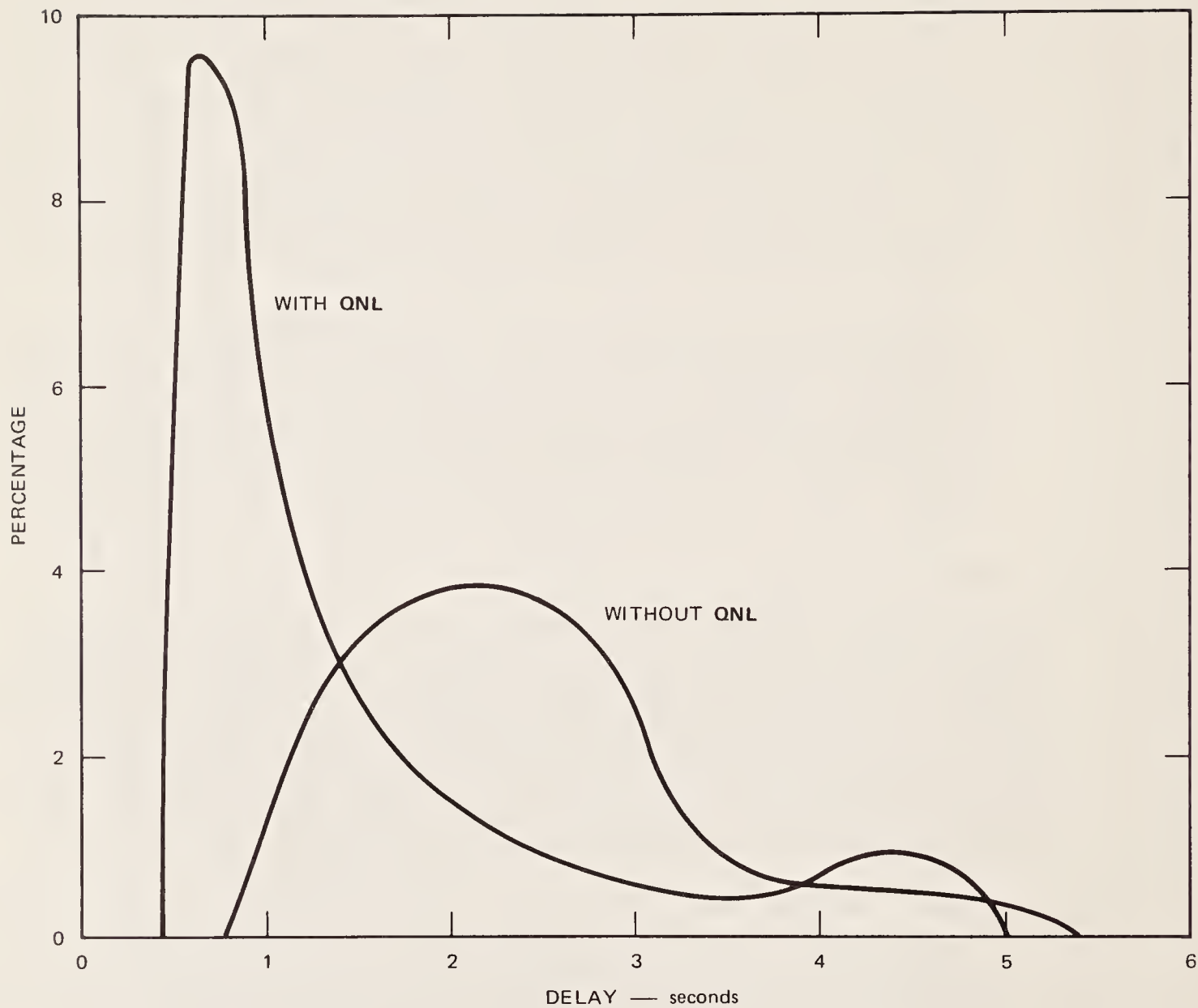
d. Addition of the QNL Queue

The simulation was rerun with the addition of a special queue (QNL) for interactive users. This queue has the effect of assigning a higher priority to highly interactive functions, at the expense of other tasks. Figure IV-3 shows the (approximate) distributions of delay times for NLS file-reference tasks with and without QNL, when the system is serving 3 NLS users, 3 TODAS users, and 1 OTHER user. The improvement resulting from the use of QNL is clear.

With respect to Fig. IV-3, it is informative to consider what happens to the single program of type OTHER in this situation. It was expected that the use of QNL would result in slowing the OTHER program; however, the actual effect was a slight increase in its execution speed.

This is caused by a decrease in swapping in the system when QNL is used. Since interactive jobs are reactivated more quickly, there is a greater chance of needed pages still being in memory, thus reducing the swapping. The overall effect is an increase in system efficiency.

In general, however, the use of QNL may result in a



TA-7101-8

FIGURE IV-3 SYSTEM WITH AND WITHOUT QNL: DISTRIBUTION OF DELAY TIMES (IN SECONDS) FOR NLS FILE-REFERENCE TASKS—3 NLS USERS, 3 TODAS USERS, 1 OTHER USER

slowing of OTHER programs. During a given interval of time, the programs for OTHER users take up all the system resources that are not used by NLS or TODAS users. When QNL is included in the scheduling algorithm, NLS and TODAS users are able to get better response and thus they work faster, taking up more of the system's resources during a given interval. Thus if there is a large number of interactive tasks, the programs of type OTHER will receive less time.

e. Drum Access and Bandwidth

It is apparent from Fig. IV-2 that the major factors affecting response time are the delay encountered in swapping and, to a lesser extent, file input/output. The obvious way of improving this situation is to provide a device with higher bandwidth for swapping and file input/output.

In this study we have not attempted to present general results relating response to these factors. Instead, we have taken as a specific example a particular drum that could replace the present drums used with the 940 system.

The current drums have a rotation time of 34 milliseconds and a transfer time of about 17 milliseconds for a 2K page of 24-bit words. The drums used for comparison have a rotation time of 8.5 milliseconds and a transfer time of about 5.7 milliseconds per page.

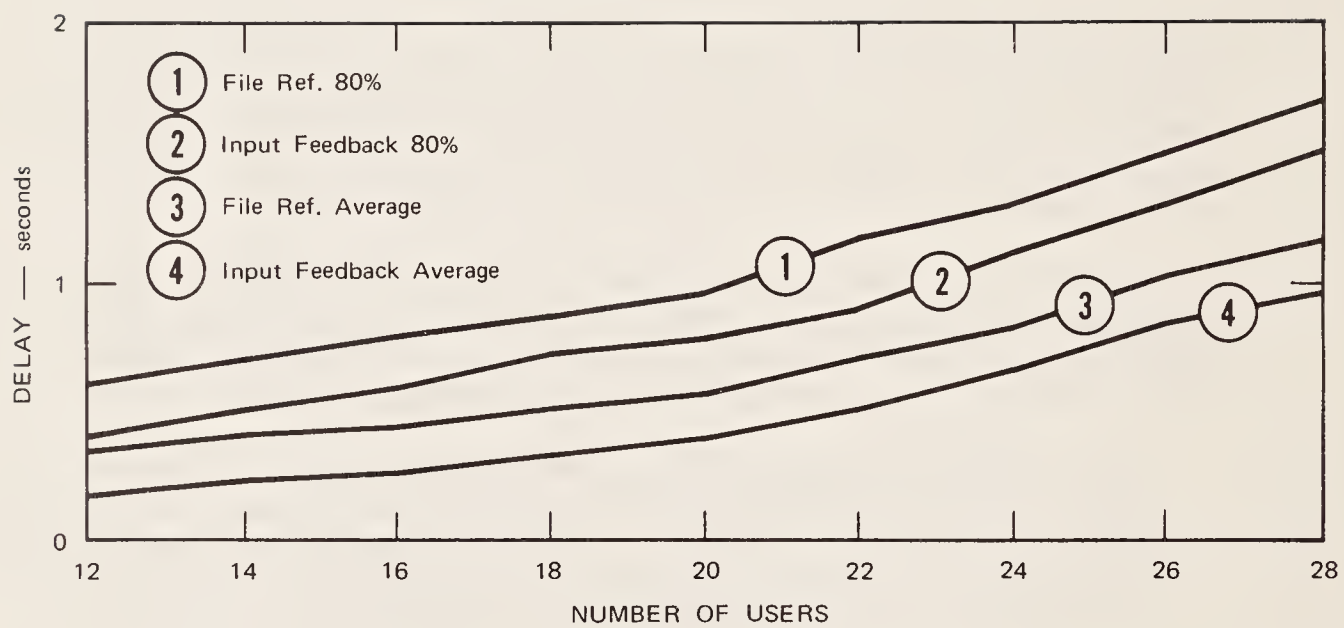
In addition, the new drums will allow a page transfer to begin at any point. This means that the average time to read or write a page will be approximately equal to the duration of a single revolution.

The effect of the new drums as predicted by the simulation is very striking.

A large part of this is due to the consistent completion of interactive tasks within a short quantum. With slower drums these tasks often take several short quanta.

Figure IV-4 shows the average and the 80-percent times for NLS input-feedback and file-reference tasks for a system with QNL, one OTHER user, and the remaining users evenly divided between NLS and TODAS.

Notice that the difference between the categories remains



TA-7101-9

FIGURE IV-4 SYSTEM WITH QNL AND NEW DRUMS: AVERAGE AND 80-PERCENT TIMES FOR NLS INPUT-FEEDBACK AND FILE-REFERENCE TASKS WITH 1 OTHER USER AND REMAINING USERS EVENLY DIVIDED BETWEEN NLS AND TODAS

relatively small and constant. This is because both are being consistently completed within a single activation, so that the difference in elapsed time is simply the difference in time required to do the actual task.

As the number of users increases, the delays increase because of longer queues. Thus the limiting factor with the faster drums will be congestion in the queues and resulting delays for input-feedback tasks, rather than the delays for file-reference tasks, as is the case in the current system.

f. Speed of Central Processor

In view of the very small percentage of time spent doing computation, it is interesting to consider the effect of varying the speed of the central processing unit (CPU).

Figure IV-5 shows the 80-percent time for NLS file-reference tasks with the current system and CPU's of various speeds.

The difference is small even with a range of 400 to 1 for CPU speeds. Clearly, improvement that will benefit a system such as NLS should be sought elsewhere than the CPU.

g. Size of Core Memory

Although the XDS940 is limited to 64K of 24-bit words for core memory, it is interesting to study the effect of adding more core.

Figure IV-6 shows the 80-percent times for NLS file-reference tasks with the current system and various sizes of core memory.

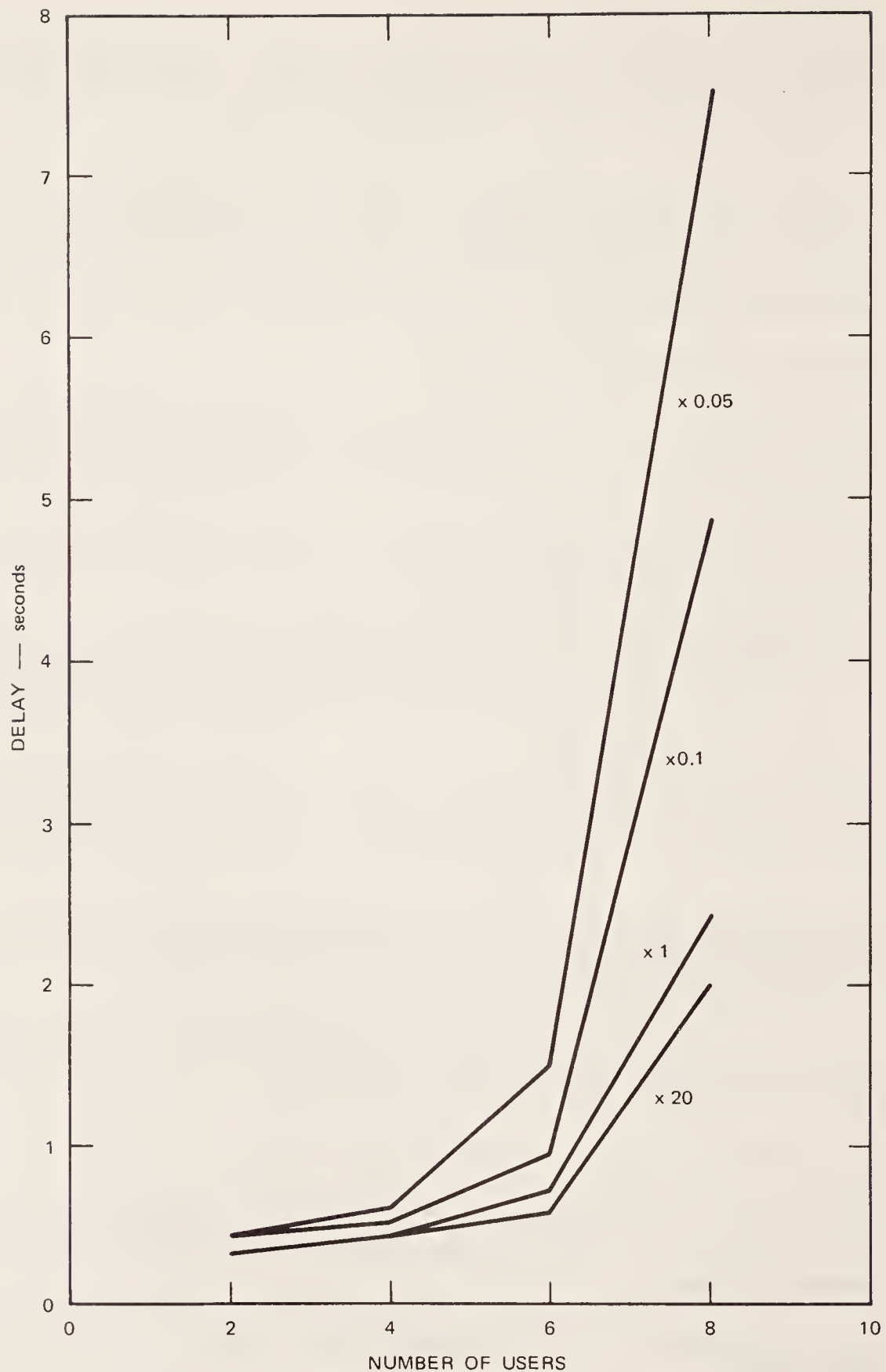
These results should be considered only as lower bounds, since different scheduling algorithms could be expected to make better use of a larger memory.

h. Interactive Display Subsystem (IDS)

From the above discussion, it is clear that the greatest improvement in system responsiveness results from the use of faster drums.

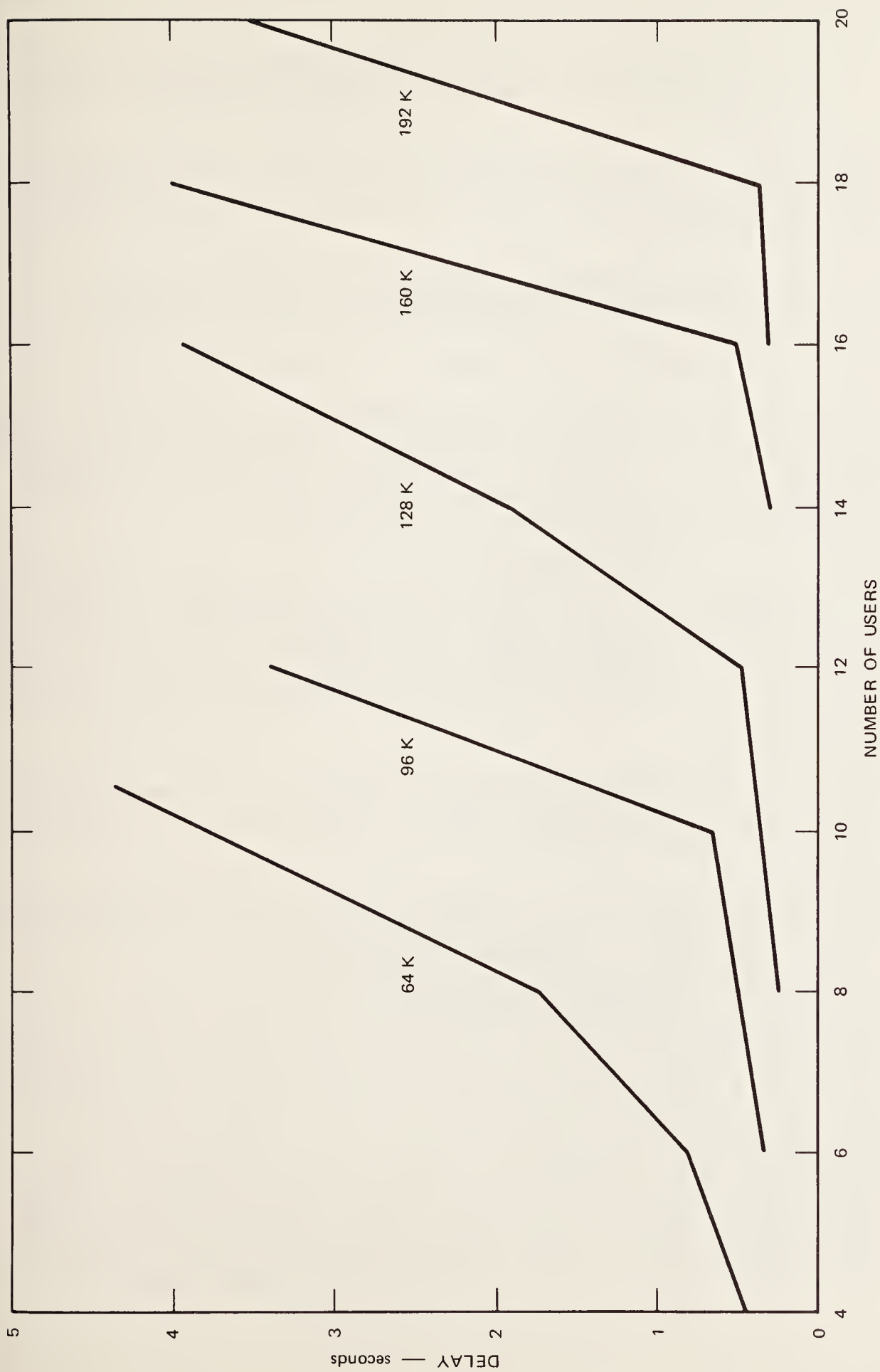
The limitations of the system with new drums are the following:

- (1) Long queue lengths resulting in poor response for



TA-7101-10

FIGURE IV-5 CURRENT SYSTEM WITH VARIOUS CPU SPEEDS
RELATIVE TO CURRENT SYSTEM CPU: 80-PERCENT
TIMES FOR NLS FILE-REFERENCE TASKS—USERS
EQUALLY DIVIDED BETWEEN NLS AND TODAS



TA-7101-11

FIGURE IV-6 CURRENT SYSTEM WITH VARIOUS CORE SIZES: 80-PERCENT TIMES FOR NLS
FILE-REFERENCE TASKS—USERS EQUALLY DIVIDED BETWEEN NLS AND TODAS

input-feedback tasks

- (2) Decreasing number of available pages as number of NLS users increases (because of pages needed for display buffers).

The interactive display subsystem (IDS) is proposed as a possible solution to these limitations. It is made up of the following:

- (1) A separate core memory for display buffers so that the number of available pages remains constant
- (2) A separate processor to perform input-feedback tasks

A single input-feedback "miniprocessor," executing resident code, should be able to service a large number of NLS and TODAS users. This has the effect of giving virtually instantaneous response for input feedback, as well as reducing the load on the main processor.

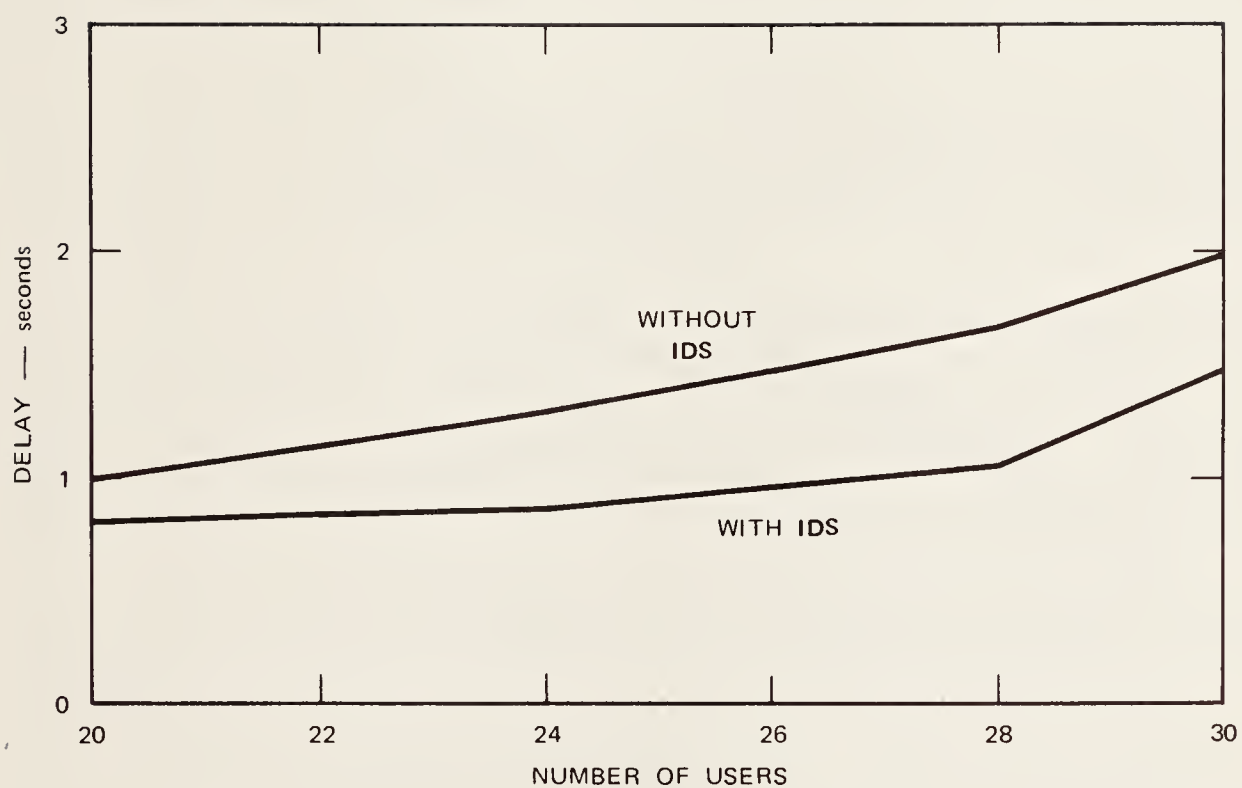
Since input-feedback tasks are by definition independent of the contents of the file currently being referenced, the miniprocessor needs only a small description of the current command state of the user. Feedback is the same for all users, so a single program will suffice. This program will be resident in the separate core, so swapping will not be necessary.

When a user calls for the execution of a file-reference task, the miniprocessor passes identifying information to the main processor.

This approach should be applicable to any timesharing system that is concerned with servicing a large number of users for a small number of interactive programs.

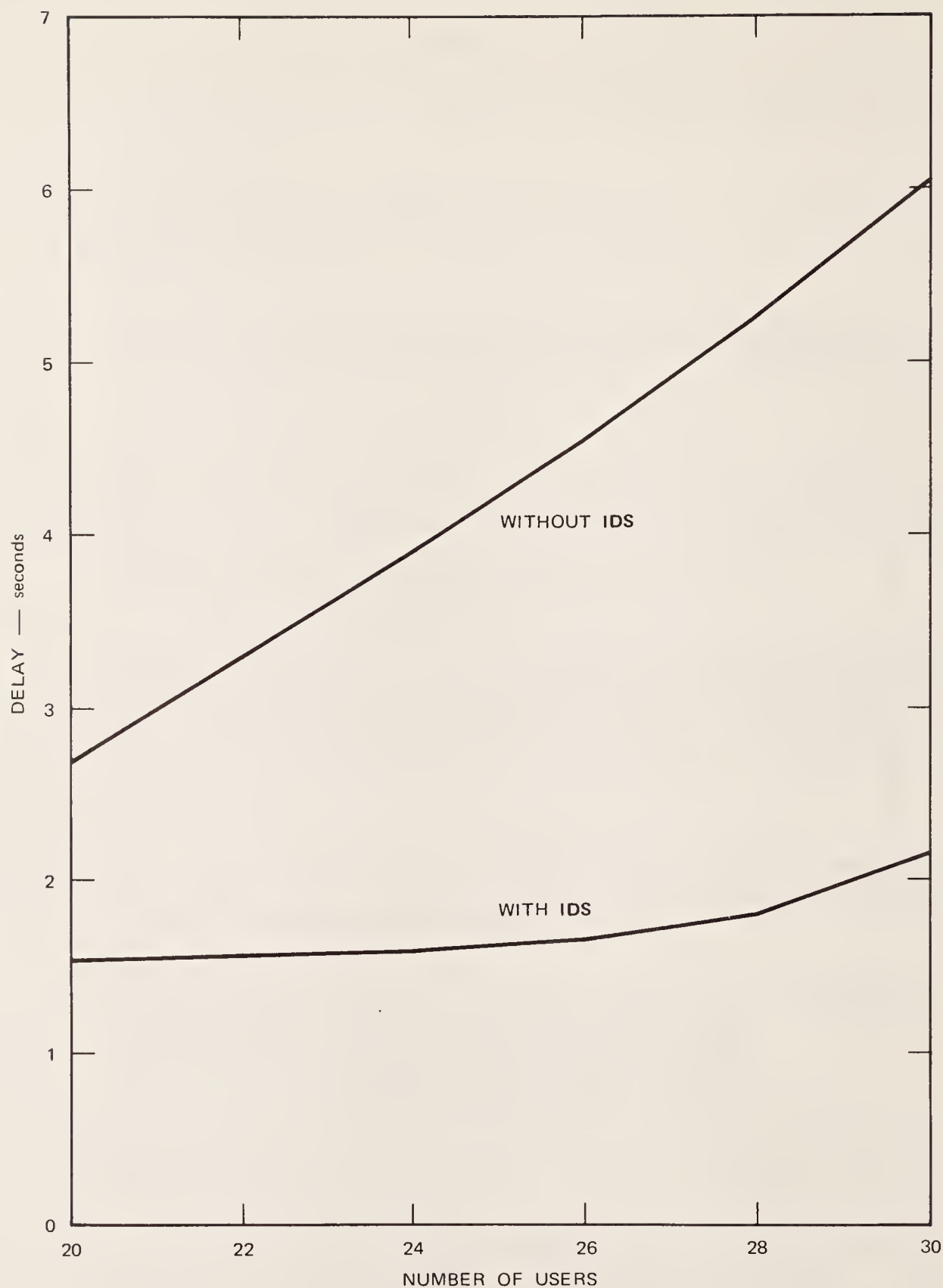
Figure IV-7 shows the 80-percent delay for NLS file-reference tasks in a system with QNL and new drums, with and without IDS. There is one OTHER user; the remaining users are equally divided between NLS and TODAS.

The minimum total elapsed time for a simple editing operation shows the value of IDS more vividly. (An "operation" here means the sequence of actions that an NLS user goes through to achieve some desired effect; the sequence typically includes several actions that require input feedback and one that requires file reference.)



TA-7101-12

FIGURE IV-7 SYSTEM WITH QNL AND NEW DRUMS, WITH AND WITHOUT IDS: 80-PERCENT TIMES FOR NLS FILE-REFERENCE TASKS—1 OTHER USER, REMAINING USERS EQUALLY DIVIDED BETWEEN NLS AND TODAS



TA-7101-13

FIGURE IV-8 SYSTEM WITH QNL AND NEW DRUMS, WITH AND WITHOUT IDS: 80-PERCENT TIMES FOR SEQUENCE OF 3 INPUT-FEEDBACK TASKS AND 1 FILE-REFERENCE TASK—1 OTHER USER, REMAINING USERS EQUALLY DIVIDED BETWEEN NLS AND TODAS

Figure IV-8 shows the total 80-percent delays for a sequence of three input-feedback tasks and one file-reference task, in the same system configurations as shown in Figure IV-7.

With IDS, input-feedback tasks may be assumed to be completed in a quarter of a second (for the numbers of users considered). The curves of Figure IV-8 show the resulting dramatic improvement in service to the user.

E. The On-Line System, NLS

1. Introduction

NLS, as currently implemented, is a highly sophisticated text-manipulation system oriented toward on-line use with displays. Its use as an augmentation tool is discussed in Appendix A.

The program is a subsystem of the timesharing system described above. Its size is currently about thirty thousand machine instructions, of which about half make up the most frequently used portions. The source languages used are MOL940 and a collection of special-purpose languages (SPLs) for command specification, content analysis, and string manipulation.

This section contains an overview of the organization of NLS, a discussion of the relationship of NLS to the 940 timesharing system, and a brief discussion of possible future developments in the program.

Appendix D contains a more detailed description of the program and the languages.

2. Overview

a. Introduction

The following is a conceptual overview of the internal organization of NLS. It is conceptual in that the overlay structure, forced upon NLS by the limited address space and fixed page size of the 940, does not always correspond to this description. Although efficiency considerations have entered into the actual implementation of NLS, the following conceptual description may still be used. It represents the design philosophy that guided the implementation, and that philosophy was followed whenever practicable.

b. Logical Organization of NLS

There are three logical levels to NLS (see Fig. IV-9).

(1) The command specification level is the highest control level. It does command recognition and handles the specification of actual operands. This is the interactive part of NLS -- the part with which a user always communicates. This level of the system is written in the input-feedback SPL.

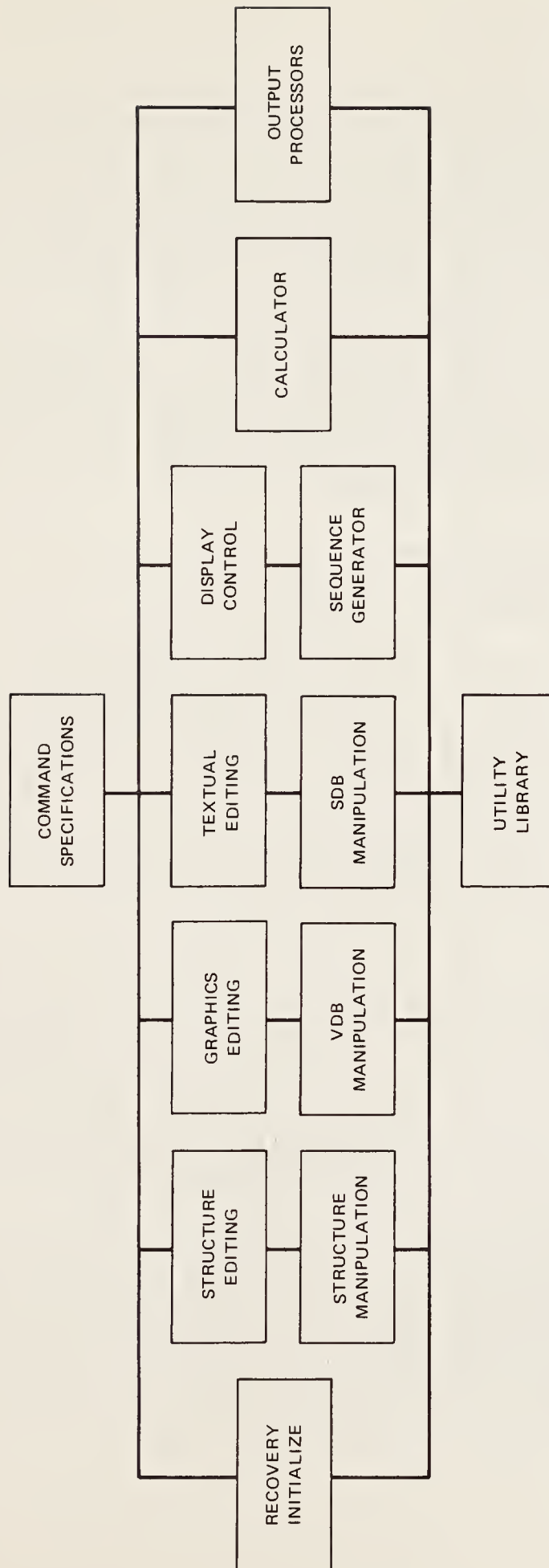
(2) The second level of control is the command algorithm level. It contains the algorithms for performing the various commands. Large parts of this level of the system are written in the content-analysis and string-construction SPLs.

(3) Utility routines make up the third and lowest level of control. These are the routines that actually change the data base, perform I/O, etc. Each of these routines is used by several routines on the second level and sometimes by the first level. The utility routines are the only part of NLS that is significantly dependent on the hardware, operating system, or data structure. The higher levels are all algorithms written with little or no consideration for the environment in which they operate. This lowest level of the system is written in MOL.

Command Specification Level

The command specification part of NLS takes input from the user to determine what command is to be executed and the actual operands for the operation. It then transfers control to the appropriate place in the second level to execute the command. Thus, this is the level where commands and actual operands are specified, but no actual execution of the commands is done.

The command specification algorithm of NLS is implemented as a large set of nested case statements. The code gets an input character and tests it in a case statement, which results in some feedback to the user and transfer of control to the head of another case statement to test the next character of input.



TA-7101-5

FIGURE IV-9 LOGICAL ORGANIZATION OF NLS

Command Algorithms

The second level of control consists of the code that implements the algorithms for the various commands. This level consists primarily of calls on utility routines that access the data files, test the data elements to determine exactly what should be done, and call on the appropriate utility routines to perform the actions required by the command being executed.

The command algorithm code has been organized into several divisions based on the commands they effect. The code for each division of commands is further divided into a part that includes the algorithms proper and a part that is more related to (and thus dependent on) the logical data structure.

There are eight main divisions:

(1) Structure Editing

NLS files have a ring structure. Each element in the ring represents a statement and its associated character string and/or line drawing. The character string itself is stored in a statement data block (SDB), while the line drawing is stored in a vector data block (VDB). Each ring element contains pointers to its associated SDB and VDB as well as the information that determines its position in the ring.

There is a full set of editing commands that involve the manipulation of the ring structure alone and do not alter the contents of the statements (e.g., the "Move Statement" command). The algorithms for these commands are in this section. They are independent of data structure and use the structure-manipulation machinery to actually effect changes in the file.

The structure (ring element) manipulation section contains the algorithms for altering ring elements in order to effect structure editing. They are dependent on the logical data structure, but not on the physical data structure (utility routines are used to actually change the physical data).

(2) Text Editing

This section contains the algorithms for doing editing on the text of statements, e.g., the "Insert Word" command. These algorithms are independent of data structure. They use the content-analysis machinery to determine where changes should take place, and the string-manipulation and SDB-manipulation machinery to actually effect changes to the file (through the use of utility routines).

The content-analysis section (used for locating textual patterns within a string) and the string-manipulation section are independent of the physical and logical structures of the file.

The SDB manipulation section, used for altering SDB blocks, is not dependent on the physical data structure but is dependent on the logical data structure.

(3) Graphics Editing

This section contains the algorithms for commands that edit line drawings (e.g., the "Insert Vector" command), and is independent of the logical and physical structures of the data. This code uses the VDB manipulation machinery to effect changes to the file.

The VDB manipulation section, used for altering VDB blocks, is dependent on both the logical data structure and the internal representation of vectors.

(4) Display Control

NLS has an assortment of controls that permit a user to specify which statement is to be displayed at the top of the screen (the "display-start statement") and the selection processes to be used in determining which statements of the file will actually be displayed.

(a) Jump and Link Machinery

The first function is implemented in the "jump"

and "link" machinery.

The jump machinery is used to select a display-start statement. A ring of past display-start statement identifiers and associated display parameters is maintained to permit the NLS user to return to previous views of his file.

The link machinery is similar to the jump machinery, except that the new display-start statement may be in another file, in which case a link stack is used instead of the jump ring.

(b) Sequence Generator

Once the display-start statement has been determined, the sequence generator is used to select statements from the file according to currently invoked filtering criteria.

The sequence generator uses the display parameters, content analysis, and keyword reorganization when appropriate. These facilities are discussed below.

The sequence generator begins at the display-start statement and goes through the ring structure of the file, testing each statement against the filtering criteria and returning those statements that pass.

For instance, the user may have specified that he wishes to see only the first two levels of the ring structure, or only those statements which meet some criterion specified by a content-analyzer pattern (see below).

(c) Display Parameters

Display parameters controlling the selection processes of the sequence generator may be set at any point in the specification of a command.

The user also has at his disposal a set of display-format control parameters (VIEWSPECS)

for modifying his view of the file.

(d) Content Analyzer

A compiler is used to generate code from text written in a special high-level user language, and this code is used to test a statement for specified content. The content-analysis language available to the user is a subset of the content-analysis SPL mentioned earlier, which is used for other content-analysis code in the system (e.g., for delimiter identification in text-editing commands).

If content-analysis filtering is being invoked, the sequence generator uses the compiled code to test statements that have passed all of the other criteria.

(e) Keyword Reorganization

A list of statement identifiers is constructed in response to user selection and weighting of keywords (named statements containing lists of other named statements). This list is saved with the file.

If keyword reordering is being invoked, the sequence generator uses the list in generating a sequence of statements.

(f) Create Display

The set of routines called "create display" uses the display-start statement identifier, the sequence generator, and the display parameters to format and construct a display for the user.

(5) Calculator

The calculator division is a group of routines that effect arithmetic manipulations on numbers stored in an NLS file, providing the user with on-line numerical calculation capability.

(6) Processors

The processors are not part of NLS proper, but are

activated by NLS as subprocesses of NLS. They use NLS machinery -- primarily the sequence generator -- to provide input from NLS files.

Those currently implemented are the MOL compiler, the SPL compiler, the Tree Meta compiler, and the Output Processor, which formats NLS files for hardcopy output to various devices.

(7) File I/O

The file I/O division effects file loading and output.

(8) Recovery and Initialization

Routines in this section are executed when NLS is started up or continued after exiting to the timesharing executive.

Utility Routines

The utility-routine level of NLS is a collection of subroutines (written in MOL) that actually do things. In a sense the higher two levels merely decide what to do and in what order. These levels are essentially independent of the machine, operating system, file system, and physical data structure.

On the utility level, data files are changed and I/O occurs. Some of the utility routines are used by the two higher levels to read the current state of the data files. The higher levels use this information to decide what to do.

This level contains all routines that actually read or change data files, interact with the operating system, or do I/O to the work stations. In this manner all code that is dependent on the environment (hardware, software or physical data structure) gets put in one place. The advantages when moving to a new machine or when the environment changes are obvious. Another consideration is the hope that a fairly complete library of routines will be built up and the subsequent implementation of a new command should then be quite easy.

3. Relation of NLS to the XDS940 and the Timesharing System (TSS)

The most significant features of the XDS940 timesharing system that affect NLS and are used by it are programmed operators, the file system, paging, and forks.

a. Programmed Operators

Programmed operators (called "POPs") are used extensively in NLS and the compilers.

By means of a POP, a subroutine may be called just as if it were a machine instruction.

This means that the address field of the instruction may be used to pass an argument to the subroutine, resulting in higher code density.

In addition, for reentrant code, the transfer to a subroutine as a POP can be executed significantly faster than the transfer to a normal subroutine.

b. File System

It is important that the time required to carry out an operation on an NLS file not increase greatly as the file becomes larger. This requires the ability to access random segments of the file with a delay independent of the location of the segment in the file. The TSS random file system makes this possible.

Any block of information in a random file may be referenced by a system function which is given the file identification, an address in the file, an address in memory, and the number of words to be transferred as arguments.

The address space of the file is broken up into a number of blocks of fixed length (currently 256 words). Additional blocks, not in the file's address space (and hence available only to the system), are used to record the locations of the file blocks in secondary storage. The first such index block contains addresses for the first 144 blocks of addresses in the file. If higher addresses are used then additional index blocks may be used.

c. Paging Mechanism

The address space of a program on the 940 can consist of up

to eight pages of 2048 words each. This is not large enough to hold all of NLS, and necessitates a rather complex overlay structure. Before this can be explained, a brief discussion of the paging mechanism in TSS is needed.

While a program can have only eight pages in its address space at any one time, it can have up to 63 pages to choose from. These correspond to the 63 possible entries in the job's program memory table (PMT).

Pages may be made available (entered in PMT) in two ways:

- (1) When a program is first activated by the user, the (up to 8) pages making up the program are placed in the PMT.
- (2) Additional pages may be added to the PMT by the program itself.

To do this, it executes a system function with a file name as argument. The named file should contain up to eight additional pages of program.

The system enters these pages into the PMT and returns indices by which the pages may be referenced. Such an index into the PMT is called the "relabeling byte" for the page.

The relabeling for a program consists of the eight relabeling bytes for the pages currently making up the program. (Unused pages have the relabeling byte set to zero.)

A program may read and set its own relabeling by means of system functions. This allows the program to bring pages from its PMT into its address space by simply putting the appropriate relabeling bytes into its relabeling.

For a more detailed discussion of these features the reader is referred to Ref. 13.

d. Forks

The final feature of the TSS used by NLS is the ability to create independent processes (called forks) within a single job.

The particular uses of forks in NLS are discussed in

Appendix D.

h. Future Developments

The short-range extensions of NLS will include both modifications of existing features and introduction of new ones. The following is a partial list of the possibilities currently under consideration.

The graphics capability will have a wider variety of entities and editing operations.

The calculator will allow several named functions to be maintained simultaneously and will be able to produce plots.

It will be possible to split the text area into several windows, allowing multiple simultaneous views of a file. A later stage will allow different files in the windows and cross-file editing.

Tables will be introduced as special entities consisting of two-dimensional arrays of strings, with columns either left or right justified. It will be possible to display subsets of rows and columns.

Special features will be added to facilitate the use of NLS in support of on-line dialogue. These include explicit structures for backlinks and comments.

The keyword system will be replaced by a more sophisticated retrieval system, including automatic generation of inverted lists from catalogs. The user will have languages to define, store, and display sets of catalog entries.

A general interface between NLS and processors, such as compilers, will be developed.

A processor will be written which will reconstruct a file in such a way that statements that are structurally "close" will also be physically close, thus minimizing file I/O for display construction.

It will be possible to have links converted to page-number references in hard copy.

F. The ARPA Computer Network

1. History

Two prototype user-program interfaces to the ARPA Network were written, and were used in primary communications between UCLA and SRI and between SRI and the University of Utah. The first of these went into operation in late November 1969.

2. Current Status

The permanent Network operating system is now being finished, and will be operational in April 1970.

The Network monitor will be characterized by two different interfaces, one to be used by persons operating on the Network using the ARC 940, and the other to be used by programs running on the 940 and communicating with other hosts on the Network.

To a person on the Network, the 940 will initially appear (with the exception of certain procedural characteristics) as it would were he connected to it via an ordinary Teletype linkage.

The 940 monitor, after dispensing with the procedural transmissions necessary for establishing a primary link, simply reads characters from the Network and places them into the Teletype input buffer of an unattached 940 station.

In parallel with this operation, it transmits the contents of that station's Teletype output buffer over the Network.

The 940 user wishing to use another host on the Network must do so either by writing a user program which contains the necessary monitor calls or by calling a special Network subsystem (running on the 940) which interfaces to the monitor and makes the necessary calls for him.

The monitor calls are designed in such a way that the programmer may consider the Network to be an input/output device. Accordingly, calls are provided for the following functions:

(1) OPEN PRIMARY LINK

A primary link is established by calling a system

function with parameters designating the desired destination host.

When an attempt is made to open a primary link, success is indicated by a skip return and a file number (which may be used in successive transactions for identifying the link); failure is reflected by a non-skip return and an error code.

Assuming a successful return from an OPEN PRIMARY LINK request, the user may immediately begin transmitting information over the link, using the input/output functions described below.

OPEN PRIMARY LINK is a special system call which is unrelated to the other system commands for opening files.

(2) CLOSE PRIMARY LINK

CLOSE PRIMARY LINK causes the system to disconnect a primary link (identified by the file number obtained from OPEN PRIMARY LINK) after checking its validity. A failure in closing the link results in an illegal-instruction trap.

CLOSE PRIMARY LINK is a special system call which is unrelated to the the other system commands for closing files.

(3) INPUT/OUTPUT TO PRIMARY LINK

Input/output is handled in the same way as the other file I/O on the 940.

The initial Network monitor will perform single-character output over the Network. Provision has been made for multiple-character output, and it is expected to be implemented shortly after the initial Network monitor is operational.

3. Implementation

There are two basic tasks for which the Network monitor must be responsible: the provision of the I/O drivers necessary for using the Network, and the development of a protocol for host-host communication.

The I/O drivers have such functions as the following:

- (1) Initiation of input/output commands to the hardware interface
- (2) Detection of hardware interface errors and execution of proper corrective or evasive actions
- (3) Buffer allocation and manipulation
- (4) Correct formatting of messages so far as the IMPs and the Network are concerned
- (5) Detection of IMP/Network errors and proper error action
- (6) Notification of 940 status to the IMP and Network
- (7) Initialization and recovery after 940 system crashes
- (8) Allocation and maintenance of links over the Network, including the handling of RFNMs
- (9) Maintenance of necessary internal tables pertaining to the Network
- (10) Communication between the Network and ARC 940 work stations.

This includes the basic system calls required for input/output, the manipulation of Teletype I/O buffer when a remote user is connected to the 940 as a telephone-line type user, notification of work stations about Network errors, notification of work stations about illegal requests, etc.

A protocol has been established which hosts must adhere to in order to communicate effectively.

The monitor must be able to respond to this protocol in order to use the Network.

Although the protocol is not yet in final form, some of the probable areas of concern will be:

- (1) Opening and closing of primary links
- (2) Opening and closing of auxiliary (file-transfer)

links

- (3) Message formatting (host-host)
- (4) Control message decoding and interpretation
- (5) Communication of status.

Since the fundamental Network drivers will be static once they are implemented, they have been integrated into the existing monitor as efficiently as possible.

The protocol, however, will probably be subject to change for some time; therefore, it is being implemented in a less integrated but more flexible manner.

Among other things, it is being coded in MOL940, which will make it easier to debug and modify than if it were coded in assembly language.

The general implementation approach is to a large extent dictated by the space restrictions in the 940 monitor.

We have tried to put as little code as possible in the resident monitor pages, and as much as possible in a separate page which may be relabeled in and out of the monitor's relabeling.

Thus the resident routines in the monitor are mainly the ones that are necessary for processing interrupts and certain communications (there are cases when the Network code must communicate with another page which runs in the same position). The remainder of the Network code, and buffer space, resides in the separate page.

G. The NLS UTILITY Subsystem

Manipulation of the large number of files which are directly used in connection with compiling, assembling, loading, and debugging NLS is a significant problem. Accordingly, a subsystem called "NLS UTILITY" has been written to help handle these files.

NLS UTILITY performs the functions described below for the symbolic, binary, and core-image files of NLS and PASS4 (the output processor).

1. Archiving

All files relating to NLS are permanently stored on the disc under an archiving system.

In order for the files to be accessed, they must be explicitly read from the archives to temporary storage, and any permanent changes to a file must be recorded by writing the updated version of the file from temporary storage to archive storage.

NLS UTILITY performs these functions for the user, as well as ensuring the integrity of files written into archival storage.

2. Compilation

Subprograms for NLS are written in three different programming languages.

The compilation process is different for different languages, and there is in some instances an interaction between one symbolic file and another.

The concern that an NLS programmer need have with the details of NLS compilation is minimized by NLS UTILITY.

With NLS UTILITY, any or all of the NLS subprograms may be compiled; the compilation results are reported to the user in a manner which he designates.

3. Loading

The loading process for NLS is somewhat complex.

The unloaded NLS system consists of more than 50 binary files, and they must be loaded in a certain order and in a certain relationship to each other.

As in compilation, NLS UTILITY makes it unnecessary for the NLS programmer to concern himself with the peculiarities of loading.

The loaded system consists of 7 core-image files.

While the files are closely related, there is frequently value in loading only one or another of them.

For this reason, NLS UTILITY allows a variety of loading options, including one which loads the entire system, and one

which loads a specific file.

4. Listing

Because of the size of NLS, the maintenance of up-to-date listings is a tedious job.

Functions provided in NLS enable the programmer to produce any number of listings of any or all NLS symbolic files by a simple process.

More details on the individual functions and the operation of NLS UTILITY may be found in Appendix D.



V FUTURE PLANS

A. General

Future directions for work in the ARC will be influenced by forces originating both inside and outside the Center.

Forces generated by our cumulative experience in the development of augmentation systems within the Center indicate some new directions for our own bootstrapped research effort.

External forces are generated by our participation in the ARPA Network experiment and by an increased awareness for the need to communicate with the "outside world" -- people outside the Center who are engaged in related work.

The internal forces and those generated by our Network participation combine to produce a shift in our internal research emphasis towards two specific activities: (1) team augmentation and (2) the development of a system design discipline. These are discussed below under "Shifts in Emphasis."

Increased awareness of the need to communicate and interact with the outside world will lead toward the development of a new area of specific concern, discussed below under "Transfer of Results."

The goals associated with research in team augmentation, with the development of a system design discipline, and with the transfer of results are related to one another within the ARC goal structure as described below in the section entitled "Short-Term and Long-Term Goals."

In the section "Selected Plans Under Other Sponsorship," we discuss the System Developer Interface Activity (SYDIA), for which we are seeking additional sponsorship. It is intended that this activity will be the primary effort in the area of the transfer of results.

B. Shifts in Emphasis

Our plans reflect a maturing shift in emphasis in our research work. We plan to shift our emphasis toward two basic activities: (1) team augmentation and (2) the development of a system design discipline.

1. Team Augmentation

Whereas in the past we have given most of our attention to augmenting the individual worker, we are now focussing on the augmentation of a team of collaborating workers, each of whom is individually augmented.

The high mobility and manipulative capability of a skilled "augmented individual" has a unique potential which can be realized when a number of augmented individuals join into a collaborative team. Not only can each individual move very rapidly through the joint working files to study them, enter new information, and update old material, but this power can be amplified by special computer aids, conventions, and skills that directly facilitate the processes of intercommunication and coordination.

The contemplated efforts in "team augmentation" involve several facets:

(1) The development of conventions and procedures for organizing the working records of our plans, designs, objectives, design principles, schedules, etc., so as to give effective mutual "task orientation" to the members of a team by ensuring optimal accessibility of all information related to the team's objective.

(2) The special development of a "Dialogue Support System" to facilitate the rapid evolution of these working records via dialogue among members of the design team.

(3) The development of techniques to facilitate simultaneous remote collaboration among people at physically remote on-line terminals (of any sort), by giving them direct communication with one another, independent of their current individual work interactions with the computer. This includes provision, where feasible, for the following:

(a) Video and/or voice intercommunication

(b) Easy and flexible control of means for duplicating, at any terminal, all or part of the type-out or display from another terminal

(c) Ready transfer of control of one terminal's computer interaction to another terminal's input

devices.

These techniques will evolve within ARC under conditions of application to our own coordinated system-development work, and will be applied over a wide range of collaborative actions, from simple question-answering facilities to complex design work involving intense mutual participation by the team members.

As applicable techniques become effective within ARC, we will explore their use and value for the following:

- (1) Support of Network Information Center (NIC) services such as teaching, question-answering, and some types of query servicing
- (2) Working collaboration between ARC staff and personnel at other Network sites
- (3) Working collaboration between people at remote Network sites, independent of ARC staff.

2. Development of User- and Service-System Design Discipline

The functional features of the "user system" -- the large collection of computer aids available to an ARC worker -- have evolved with some ingenuity, a great deal of cut-and-try experimentation under actual-usage conditions, and a certain special orientation offered by our overall research framework. However, up to now there has been a significant lack of objective, methodical engineering design for the overall user system.

A user-system design discipline is definitely needed, and we intend to devote an increasing amount of effort toward developing such a discipline.

Like the user system, the "service system" -- the hardware and software underlying the features for augmenting users -- has evolved in an ad hoc fashion.

Here there is also a significant need for a system-design discipline.

A system-design discipline would have a communicable, teachable, generally applicable framework supporting a coordinated set of concepts, terminologies, principles, methods, and special tools.

C. Transfer of Results

Behind these basic aspects of our work in the ARC (team augmentation and design disciplines) lies an essential feature of our long-term strategy, namely, the goal of producing results that will be of direct value to other groups of system developers -- in particular, to those who will be developing augmentation systems.

This is in contrast to being of direct value to customers who will want systems for their own direct use (e.g., to augment a manager, a designer, an editor, or a researcher).

Display terminals, communication channels, and computer service are destined to become both cheap and plentiful, and it is certain that a very large number of organizations will want to use them. They must rely upon system developers who will need to be capable of the following:

- (1) Analysis of system-usage environments
- (2) Design and implementation of a smooth, powerful, and coordinated system of user aids, conventions, methods, etc.
- (3) Training and "education" of new users, many of whom will be completely unfamiliar with the potential of this new technology
- (4) Subsequent monitoring of user performance so as to implement the changes necessary to track the evolution of users' attitudes, concepts, skills, usage habits, and wants.

Although it is important to stimulate the eventual customers for augmentation systems, and to make them aware of the potential for these systems in their work, we feel that our results should be directed primarily toward helping system developers. Over the longer term, we plan to do this by pursuing the following goals:

- Item 1: Making visible an advanced, integrated system, operating in a heavy-usage environment, that can orient system developers to the available cost-value tradeoffs
- Item 2: Developing an effective system-design discipline to aid in developing augmentation systems, whether or not these systems resemble ours
- Item 3: Maintaining thorough, highly current, comprehensive documentation, designed for quick location of relevant material
- Item 4: Establishing broad-band communication channels over

which a dynamic interchange of information can take place, so that a maximum proportion of our knowledge can be quickly available in useful form

Item 5: Offering, as a model, a complete prototype design of an augmentation system especially designed for augmenting system development.

This system would be compatible with the system-design disciplines described above, and would include techniques for planning, analyzing, designing, programming, debugging, documenting, and teaching.

D. Short-Term and Long-Term Goals

Our approach to the planned work will be as follows:

(1) Achieve the short-term goals implicit in the team augmentation activity, in the development of a system design discipline, and in the tasks itemized under Transfer of Results (Section V-C above)

(2) Contribute to the long-term goal of directing our results for maximum benefit to future developers of augmentation systems.

There is considerable overlap between short-term and long-term goals.

For instance, in the case of the transfer of results, the basic bootstrapping development of techniques within the ARC seems to guarantee a very good basic buildup toward Items 1, 2, 3, and 5 of Section V-C; our participation in the Network experiment contributes directly to Item 4; and the development of the NIC service will contribute toward Items 1 and 4.

E. Selected Plans Under Other Sponsorship

To pursue directly the itemized long-range goals of Section V-C, we currently have other plans under consideration, coordinated with those outlined in this proposal. These plans would be carried out under other sponsorship:

We are formulating plans for what we tentatively call the System Developer Interface Activity (SYDIA). We expect to be approaching representative candidates during 1970 with proposals for multiple sponsorship. The initial purpose of the SYDIA will be to develop the following:

Sec. V
FUTURE PLANS

(1) A facility for an effective interchange of information skills, orientation, etc. between ARC and the existing and potential community of augmentation-system developers

(2) The ability to assist other groups to transfer our system, or parts of it, directly into another hardware environment.

Later, with specific individual funding arrangements, we would expect to begin developing close interchange relationships with various system-development groups; hopefully, some groups would then adopt our augmented techniques for system-development work.

GLOSSARY

ARC: Acronym for the Augmentation Research Center at Stanford Research Institute.

ARPA: Acronym for the Advanced Research Projects Agency.

Augmentation: Used in this report to indicate the extension of human intellectual and organizational capabilities by means of close interaction with computer aids and by use of special procedural and organizational techniques designed to support and exploit this interaction.

Center: Another term used for the ARC.

Console: As used here, this means specifically a user's control console for the ARC's On-Line System (NLS). The consoles presently in use consist of a display screen, a keyboard, a "mouse", and a "keyset."

File: As used here, this refers to a unified collection of information held in computer storage for use with the On-Line System (NLS) or with TODAS. A file may contain text (natural language or program code), numerical information, graphics, or any combination of these. Conceptually, a file corresponds roughly to a hard-copy document.

GENIE: Project GENIE, at the University of California at Berkeley, developed (under ARPA sponsorship) the timesharing software for the XDS940 computer used by the ARC.

GODOS: Acronym for Graphics-Oriented Document Output System, a means for converting NLS/TODAS files to microfilm. GODOS is capable of handling the line drawings produced with the NLS graphics capability.

IMP: Acronym for Interface Message Processor, a component used in the ARPA Network.

Keyset: A device consisting of five keys to be struck with the left hand in operating the On-Line System (NLS).

MOL: See MOL940.

MOL940: A machine-oriented language for the XDS940 computer. MOL940 (or simply MOL) was developed at ARC.

Mouse: A device operated by the right hand in using the On-Line System (NLS). The mouse rolls freely on any flat surface, causing a cursor spot on the display screen to move correspondingly.

NASA: National Aeronautics and Space Administration.

GLOSSARY

Network: The planned Advanced Research Projects Agency network of research computer installations.

NIC: The Network Information Center, to be incorporated in the ARPA network. The NIC will operate as a computer-assisted library service for information pertaining to the network, to be used by network members, and will be operated by ARC.

NLS: See On-Line System.

On-Line System (NLS): This is the ARC's principal and central development in the area of computer aids to the human intellect. As presently constituted, it is a display-oriented, timeshared, multiconsole system for the composition, study, and modification of files (see definition of "file"). A counterpart system, TODAS, operates from hard-copy terminals such as Teletypes and offers many of the same capabilities as NLS.

PASS4: An output-processing program used to convert NLS/TODAS files to hard-copy format for output via one of a number of different devices.

RADC: Acronym for Rome Air Development Center.

SPL: Acronym for Special-Purpose Language. Specifically, this term is used for the SPL's developed at ARC for use in programming NLS.

SRI: Acronym for Stanford Research Institute

Statement: The basic structural unit of an NLS/TODAS file. A statement consists of an arbitrary string of text, plus graphic information. A file consists of a number of statements in an explicit hierarchical structure.

TODAS: Acronym for the Typewriter-Oriented Documentation-Aid System. TODAS is a counterpart of NLS designed to operate from hard-copy terminals such as Teletypes.

Tree Meta: A compiler-compiler system developed at ARC.

TSS: Acronym for Time-Sharing System. Specifically, the system developed by Project GENIE for the XDS940 computer.

XDS940: The computer facility used by ARC is based upon a Xerox Data Systems (formerly Scientific Data Systems or SDS) model 940 timesharing computer.

940: See XDS940.

REFERENCES

The following is a list of references specifically cited in the report.

1. D. A. Evans, "Man/Computer Augmentation Systems for Qualitative Planning," Ph.D. Thesis, Department of Civil Engineering, Stanford University, Stanford, California (December 1969).
2. "Specifications for the Interconnection of a Host and an IMP," Report No. 1822, Contract No. DAH015-69-C-0179, ARPA Order No. 1260, Bolt Beranek and Newman Inc., Cambridge, Massachusetts (May 1969).
3. D. C. Engelbart, W. K. English, and J. F. Rulifson, "Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research," Final Report, Contract AF 30(602)4103, SRI Project 5919, Stanford Research Institute, Menlo Park, California (April 1968), AD 843 577.
4. D. C. Engelbart, W. K. English, and D. A. Evans, "Study for the Development of Computer Augmented Management Techniques," Interim Technical Report RADG-TR-69-98, Contract F30602-68-C-0286, SRI Project 7101, Stanford Research Institute, Menlo Park, California (March 1969), AD 855 579.
5. D. C. Engelbart and B. Huddart, "Research on Computer-Augmented Information Management," Technical Report ESD-TDR-65-168, Contract AF 19(628)-4088, Stanford Research Institute, Menlo Park, California (March 1965), AD 622 520.
6. W. K. English, D. C. Engelbart, and B. Huddart, "Computer-Aided Display Control," Final Report, Contract NAS1-3988, SRI Project 5061, Stanford Research Institute, Menlo Park, California (July 1965), CFSTI Order No. N66-30204.
7. D. C. Engelbart, W. K. English, and J. F. Rulifson, "Study For The Development of Human Intellect Augmentation Techniques," Interim Progress Report, Contract NAS1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1967).
8. D. C. Engelbart, W. K. English, and J. F. Rulifson, "Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research," Final Report, Contract AF 30(602)4103, SRI Project 5919, Stanford Research Institute, Menlo Park, California (April 1968), AD 843 577.
9. D. C. Engelbart, "Human Intellect Augmentation Techniques," Final Report, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (July 1968), CFSTI Order No. N69-16140.

REFERENCES

10. W. W. Lichtenberger, "ARPAS: Reference Manual for Time Sharing Assembler for SDS 930," Document No. R-26, Office of Secretary of Defense, Advanced Research Projects Agency, Washington 25, D. C. (Revised 24 February 1967).
11. R. House, D. Angluin, and L. P. Baker, "Reference Manual for NARP, an Assembler for the SDS 940," Document No. R-32, Office of Secretary of Defense, Advanced Research Projects Agency, Washington 25, D. C. (Revised 21 November 1968).
12. R. E. Hay and J. F. Rulifson, "MOL940: A Machine-Oriented ALGOL-Like Language for the SDS 940," Technical Report 2, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (April 1968).
13. R. W. Watson, "Introduction to Time-Sharing Concepts," Technical Progress Report No. 249-68, Project No. 78140, Shell Development Co., Emeryville, California (January 1969).

BIBLIOGRAPHY

The following is a chronological list of documents published by the Augmentation Research Center.

D. C. Engelbart, "Special Considerations of the Individual As a User, Generator, and Retriever of Information," Paper presented at Annual Meeting of American Documentation Institute, Berkeley, California (23-27 October 1960).

D. C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Research Institute, Menlo Park, California (October 1962), AD 289 565.

D. C. Engelbart, "A Conceptual Framework for the Augmentation of Man's Intellect," in Vistas in Information Handling, Volume 1, D. W. Howerton and D. C. Weeks, eds., Spartan Books, Washington, D.C. (1963).

D. C. Engelbart, "Augmenting Human Intellect: Experiments, Concepts, and Possibilities," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Research Institute, Menlo Park, California (March 1965), AD 640 989.

D. C. Engelbart and B. Huddart, "Research on Computer-Augmented Information Management," Technical Report ESD-TDR-65-168, Contract AF 19(628)-4088, Stanford Research Institute, Menlo Park, California (March 1965), AD 622 520.

W. K. English, D. C. Engelbart, and B. Huddart, "Computer-Aided Display Control," Final Report, Contract NAS1-3988, SRI Project 5061, Stanford Research Institute, Menlo Park, California (July 1965), CFSTI Order No. N66-30204.*

W. K. English, D. C. Engelbart, and M. L. Berman, "Display-Selection Techniques for Text Manipulation," IEEE Trans. on Human Factors in Electronics, Vol. HFE-8, No. 1, pp. 5-15 (March 1967).

D. C. Engelbart, W. K. English, and J. F. Rulifson, "Study For The Development of Human Intellect Augmentation Techniques," Interim Progress Report, Contract NAS1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1967).

J. D. Hopper and L. P. Deutsch, "COPE: An Assembler and On-Line-CRT Debugging System for the CDC 3100," Technical Report 1, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1968).

BIBLIOGRAPHY

R. E. Hay and J. F. Rulifson, "MOL940: A Machine-Oriented ALGOL-Like Language for the SDS 940," Technical Report 2, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (April 1968).

D. C. Engelbart, W. K. English, and J. F. Rulifson, "Development of Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research," Final Report, Contract AF 30(602)4103, SRI Project 5919, Stanford Research Institute, Menlo Park, California (April 1968), AD 843 577.

D. C. Engelbart, "Human Intellect Augmentation Techniques," Final Report, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (July 1968), CFSTI Order No. N69-16140.*

D. C. Engelbart, W. K. English, and D. A. Evans, "Study for the Development of Computer-Augmented Management Techniques," Quarterly Progress Report 1, Contract F30602-68-C-0286, SRI Project 7101, Stanford Research Institute, Menlo Park, California (October 1968).

D. C. Engelbart and W. K. English, "A Research Center for Augmenting Human Intellect," in AFIPS Proceedings, Vol. 33, Part One, 1968 Fall Joint Computer Conference, pp. 395-410 (Thompson Book Co., Washington, D.C., 1968).

D. C. Engelbart and Staff of the Augmented Human Intellect Research Center, "Study for the Development of Human Intellect Augmentation Techniques," Semiannual Technical Letter Report 1, Contract NAS 1-7897, SRI Project 7079, Stanford Research Institute, Menlo Park, California (February 1969).

D. C. Engelbart, W. K. English, and D. A. Evans, "Study for the Development of Computer Augmented Management Techniques," Interim Technical Report RADG-TR-69-98, Contract F30602-68-C-0286, SRI Project 7101, Stanford Research Institute, Menlo Park, California (March 1969), AD 855 579.

D. C. Engelbart and Staff of the Augmented Human Intellect Research Center, "Study for the Development of Human Intellect Augmentation Techniques," Semiannual Technical Letter Report 2, Contract NAS 1-7897, SRI Project 7079, Stanford Research Institute, Menlo Park, California (August 1969).

*Note: Reports with AD numbers are available from Defense Documentation Center, Building 5, Cameron Station, Alexandria, Virginia 22314. Items marked with an asterisk may be obtained from CFSTI, Sills Building, 5825 Port Royal Road, Springfield, Virginia 22151; cost \$3.00 per copy or 65 cents for microfilm.

I The On-Line System (NLS)

A. Introduction

NLS, as currently implemented, is essentially a highly sophisticated text-manipulation system oriented primarily toward on-line use; i.e., it is not primarily oriented toward production of hard copy, although fairly sophisticated hard-copy formatting and output are included in the system.

NLS is intended to be used on a regular, more or less full-time basis in a time-sharing environment, by users who are not necessarily computer professionals. The users are, however, assumed to be "trained" as opposed to "naive." Thus the system is not designed for extreme simplicity, nor for self-explanatory features, nor for compatibility with "normal" working procedures.

Rather, it is assumed that the user has spent considerable time in learning the operation of the system; that he uses it for a major portion of his work; and that he is consequently willing to adapt his working procedures to exploit the possibilities of full-time, interactive computer assistance.

Thus the practices and techniques developed by users for exploiting NLS are as much a subject of research interest as the development of NLS itself.

Section IV of this appendix is a glossary of special NLS/TODAS terminology.

B. Work-Station Console

The user sits at a console whose main elements are a display screen, a typewriter keyboard, a cursor device called the "mouse," and a set of five keys operated by the left hand, called the "keyset."

The screen is used for displaying text, in various formats. The top portion of the screen (approximately 1/5 of the total area) is reserved for feedback information of various kinds: the name of the user command mode currently in effect, a "register" area used for various kinds of feedback, an "echo register" which displays the last six characters typed by the user, and other items which are explained below.

The keyboard closely resembles a conventional typewriter

keyboard, with a few extra keys for special characters and control functions. It is used for typing text as content for a file and for specifying commands, which are given as two- or three-character mnemonics.

The mouse is a roughly box-shaped object, about four inches on its longest side, which is moved by the right hand. It is mounted on wheels, and rolls on any flat surface. The wheels drive potentiometers which are read by an A/D converter, and the system causes a tracking spot ("bug") to move on the screen in correspondence to the motion of the mouse.

The user specifies locations in the displayed text by pointing with the mouse/bug combination. This eliminates the need for specifying a location by entering a code of some kind. Use of the mouse is very easily learned and soon becomes unconscious.

On top of the mouse are three special control buttons, whose uses are described below.

The keyset has one key for each finger of the left hand. The keys are struck in combinations called "chords," and each chord corresponds to a character or combination of characters from the keyboard. There are 31 possible chords; beyond this, two of the buttons on the mouse may be used to control the "case" of the keyset, giving alternative meanings to each chord. There are four possible cases, for a total of 124 possible combinations.

A simple binary code is used, and has proved remarkably easy to learn. Two or three hours' practice are usually sufficient to learn the most commonly used chords and develop reasonable speed.

The keyset was developed to increase the user's speed and smoothness in operating NLS. It was found that users normally keep the right hand on the mouse, because the great majority of command operations involve a pointing action; efficient use of the keyboard, however, requires the use of both hands, and shifting the right hand (and the user's attention) to the keyboard is distracting and annoying if it must be done for each two- or three-letter command mnemonic.

Use of the keyset permits the user to keep his right hand on the mouse and his left on the keyset,

reverting to the keyboard only for entry of long strings of text (typically five or more characters).

Originally, the keyset exactly duplicated the keyboard in function; in the development of NLS, however, certain control functions have been made two-stroke operations from the keyset where they would be three- or four-stroke operations from the keyboard. Nevertheless, it is still possible to operate all of the features of NLS without using the keyset; thus the beginner may defer learning the keyset code until he has gained some degree of mastery over the rest of the system.

C. Structured Text

"Text" is used here as a very general term. A "file" of text (corresponding roughly to a "document" in hard copy) may consist of English or some other natural language, numerical data, computer-program statements, or anything else that can be expressed as a structure of character strings. Simple line drawings can also be included in a file.

All text handled by NLS is in "structured-statement" form. This special format is simply a hierarchical arrangement of "statements," resembling a conventional "outline" form.

Each statement in a file may be considered to possess a "statement number," which shows its position and level in the structure. Thus the first statement in a file is Statement 1; its first substatement is 1A, and its next substatement is 1B; the next statement at the same level as the first is Statement 2; and so forth. Statement numbers have been suppressed in printing out most of this document, but are printed out for the remainder of this section as an example.

1a3b1a Every statement also bears a "signature" which may be displayed on command. The signature is a line of text giving the initials of the user who created the statement (or modified it most recently) and the time and date when this was done.

1a3b2 A statement is simply a string of text, of any length; this serves as the basic unit in the construction of the hierarchy. In English text, statements are normally equivalent to paragraphs, section and subsection headings, or items in a list. In other types of text, statements may be data items, program statements, etc.

1a3b2a Each paragraph and heading in this document is a NLS statement. Each statement is indented according to its "level" in the hierarchy; this paragraph is a substatement of the one above, which is in turn a substatement of another statement. A statement may have any number of substatements, and the overall structure may have any number of levels.

1a3c Note that when a user creates a file, he may let all of his statements be first-level ones, i.e., 1, 2, 3, etc. In this case he will not have to consider a hierarchical structure but simply a linear list, as is found in conventional text.

1a3c1 However, many of the features of NLS are oriented to make use of hierarchy, and the benefits of these features are lost if hierarchy is not exploited.

1a3c2 This is an example of an NLS feature to which the user must accommodate his methods; however, the experience of users has been that hierarchical structure very rapidly becomes a completely "natural" way of organizing text. Many automatic features of NLS make the structure easy to use: for example, statement numbers are created automatically at all times and the user need not even be aware of them. It is sufficient, when the user creates a statement, to specify its level relative to the preceding statement.

D. Use of the System

Text manipulation is considered to involve three basic types of activity by the user: composition, study, and modification. In practice, the three activities are so intermingled as to be indistinguishable.

1. Composition

composition is simply the creation of new text material as content for a file.

In the simplest case, the user gives the command "Insert Statement" by typing "is". He then points (with the mouse) to an existing statement; the system displays a new statement number which is the logical successor, at the same level, as the statement pointed to. The user may change the level of this number upward by typing a "u" or downward by typing a "d".

NOTE: Even if no previous statement has been created,

the system displays a "dummy" statement at the top of the text-display area, and the user points to this dummy.

The user then types the text of the new statement from the keyboard. On the screen, the top part of the text-display area is cleared and characters are displayed here as they are typed. When the statement is finished, the user hits a CA (command accept) button on the keyboard or mouse, and the system recreates the display with the new statement following the one that was pointed to.

New material may also be added to existing statements by means of commands such as Insert Word, Insert Text, and others. Properly speaking, these operations are modification rather than composition, and are discussed below.

Simple line drawings may be composed and added to the file by means of the "vector package." This is discussed in another section of this report.

2. Study

The study capabilities of NLS constitute its most powerful and unusual features. The following is only a brief, condensed description of the operations that are possible.

a. Jumping

NLS files may, of course, contain a great deal more text than can be displayed on the screen, just as a document may contain more than one page of text. An NLS file is thought of as a long "scroll." The process of moving from one point in the scroll to another, which corresponds to turning pages in hard copy, is called "jumping." There is a very large family of Jump commands.

The basic Jump command is Jump to Item. The user specifies it by entering "ji", and then points to some statement with the mouse. The selected statement is moved to the top of the screen, as if the scroll had been rolled forward.

Most of the Jump commands reference the hierarchical structure of the text. Thus Jump to Successor brings to the top of the display the next statement at the same level as the selected statement; Jump to

Predecessor does the reverse; Jump to Up starts the display with the statement of which the selected statement is a substatement, and so forth.

The Jump to Name command uses a different way of addressing statements. If the first word of any statement is enclosed in parentheses, the system will recognize it as the "name" of the statement. Then, if this word appears somewhere else in the text, the user may jump to the named statement by pointing to the occurrence of the name, or by typing the name.

This provides a cross-referencing capability which is very smooth and flexible; the command Jump to Return will always restore the previous display, so that the user may follow name references without losing his place.

It is also possible to jump to a statement by typing its statement number.

b. View Control

If a file is long, it may be impossible for the user to orient himself to its content and structure or to find specific sections by jumping through it. The principal solution to this problem is provided by level control and line truncation.

Level control permits the user to specify some number of levels; the system will then display only statements of the specified level or higher. Thus if three levels are specified, only first-, second-, and third-level statements are displayed.

Line truncation permits specification of how many lines of each statement are to be displayed. Thus if one line is specified, only the first line of each statement will be displayed.

Common usage is to use the first two or three levels in a file as headings describing the material contained under each heading in the form of substatements. Thus the user may start by looking at a display showing only the first-level statements in the file, one line of each. This amounts to a table of contents.

He may then select one of these statements and jump to

it, specifying one more level. He will then see more details of the content of that part of the file. This process of "expanding the view" may be repeated until the user has found what he is looking for, at which point he may specify a full display of the text.

Users soon develop a habit of structuring files in such a way that this process will work well. As it happens, such a structure is usually a good, logical arrangement of the material, reflecting the relationships inherent in the content.

The level and truncation controls are designed so that the necessary specifications may be made with only one or two strokes of the keyboard or keyset. These controls are only the most important of a large set of view-control parameters called "VIEWSPECS." Other VIEWSPECS control a number of special NLS features affecting the display format.

c. Content Analysis

The NLS content analyzer permits automatic searching of a file for statements satisfying some content pattern specified by the user. The pattern is written in a special language as part of the file text.

Content patterns may be simple, specifying the occurrence of some word, for example. They may also be highly complex, specifying the order of occurrence of two or more strings, the absence of some text construct, conditional specifications, etc. Simple patterns are extremely easy to write; complex ones are correspondingly more difficult.

d. "Keyword" System

A "keyword statement" is a named statement which references other statements in the file by name, in a special format. The name of the keyword statement is then understood to be a "keyword" applying to the statements referenced by the keyword statement.

Suppose that a file contains a list of keyword statements. The user may study this list and select several keywords with the Keyword Select command (pointing to the keywords with the mouse).

He may specify a weight from 1 to 10 for each keyword; if no weight is specified, a weight of 1 is assumed.

When the user gives the Keyword Execute command, a searching/scoring process is executed. Each of the selected keyword statements is scanned for the names of statements that it references. Each referenced statement receives a "score" equal to the weight of the keyword. If a statement is referenced in more than one keyword statement, the scores add.

When this process is completed, NLS constructs a display picture showing only the statements that have received nonzero scores, in order of decreasing scores.

In other words, each keyword is the name of a statement that defines some category of statements in the file. When a user selects and weights keywords, he is expressing his interest in certain of these categories. NLS then displays all of the statements in these categories, beginning with the "most interesting."

Because the relationships used in this system are set up explicitly when a user writes keyword statements, the system is very flexible although not highly automated. It may be regarded as a generalized method of reordering some of the statements in a file on the basis of user-selected criteria chosen from a supplied list (the keyword statements).

Note that this reordering is on the display, not in the file proper. The file proper is not affected in any way, except that the list of selected keywords and weights is saved in the file.

This list may be displayed on command. Individual keywords may be deleted from the list or their weights changed, or the whole list can be deleted on command.

e. Link Jumping

A "link" is a string of text, occurring in an ordinary file statement, which indicates a cross-reference of some kind. It may refer to another statement in the file, or to a statement in some other file, possibly belonging to

another NLS user. The text of the link is both human-readable and machine-readable, and the command Jump to Link permits the user to point to the link with the mouse and immediately see the material referred to.

An example of a link is (Smith, Plans, Longrange:ebtng).

The first item in the link indicates that the referenced file belongs to a user named Smith; the second is the name of the file; the third is the name of a statement in the file (a statement number may also be used); and the string of characters following the colon controls the VIEWSPECS to set up a particular view of the material.

The use of interfile links permits the construction of large linked structures made up of many files, and study of these files as if they were all sections of a single document.

3. Modification

A large repertoire of editing commands is provided for modification of files. The basic functions are Insert, Delete, Move, and Copy.

These functions operate upon various kinds of text entities. Within statements, they may operate upon single characters, words, and arbitrary strings of text defined by pointing to the first and last characters.

This set of commands is not restricted to operation within one statement at a time; for example, a word may be moved or copied from one statement to another.

The editing functions also operate at the structural level, taking statements or sets of statements as operands. A number of special entities have been defined for this purpose: for example, a "branch" consists of some specified statement, plus all of its substatements, plus all of their substatements, etc. A branch can be deleted, moved to a new position in the structure, etc.

As noted above, the modification activity tends to merge, in practice, with study and composition.

E. Summary

It must be noted that NLS is not a system designed for general usage, but a specialized tool designed for a group of people working on the development of computer aids to human intellectual processes. It is for this reason, for example, that NLS is not really a text-editing system oriented toward hard-copy production, but rather something simultaneously more general and more specialized.

It is in the process of manipulating a file -- studying it, making modifications, adding new material as an integrated process lasting for minutes or hours at a time and having a continuity extending for days, weeks, or even years -- that the real benefit of NLS appears.

An NLS file tends to become an evolving entity, subject to constant modification, updating, and reevaluation. Its development may have no clearly defined endpoint. It may cease to exist as a file by being incorporated in another file, or it may eventually be abandoned; however, it will probably never be "finished" in the usual sense of the word.

Continuous use of NLS to store ideas, study them, relate them structurally, and cross-reference them results in a superior organization of ideas and a greater ability to manipulate them further for special purposes, as the need arises -- whether the "ideas" are expressed as natural language, as data, as programming, or as graphic information.

II The Typewriter-Oriented Documentation-Aid System (TODAS)

TODAS is a text-handling system designed as a "typewriter" counterpart to NLS. In principle, TODAS can be operated from a Teletype or any other sort of hard-copy terminal, including terminals linked to the 940 through acoustic couplers and ordinary telephone lines (as opposed to NLS, which requires special transmission arrangements).

The present implementation allows for the use of Teletype Models 33, 35, and 37, Terminet and Execuport terminals (the latter having a built-in acoustic coupler), and NLS display terminals.

Each of these terminals has its own character set, no two sets being exactly the same except Teletype Models 33 and 35. As a result, special-character assignments are device-dependent. A

TODAS feature allows the user to redefine characters at will to suit his immediate purposes.

The primary purpose of TODAS is for access, within the ARPA Computer Network, to the Network Information Center (NIC) operated by ARC. TODAS will give Network users access to files of information created either with TODAS or with NLS, since files created with the two systems are identical in structure and format.

TODAS has many of the same capabilities as NLS for the manipulation of text; it differs from NLS as required by the use of a "typewriter" device instead of a display. The important differences arise from the fact that TODAS has no analog cursor device to correspond to the NLS mouse.

For this reason, editing of text within a statement cannot be done by means resembling those of NLS, since all of the NLS editing operands are indicated by the user with the mouse. TODAS uses two alternative methods.

One is the TODAS "alter" command, which operates very much like the "modify" command of the QED line-editing system developed by Project GENIE at UC. "Alter" creates a new statement to replace the original one, by going through the original from beginning to end; under user control, characters are (1) copied from the old statement to the new, (2) skipped over, or (3) inserted into the new statement from the keyboard.

The other is the TODAS "substitute" command, which allows the user to specify that a certain string of characters in the statement is to be found by TODAS and replaced with another specified string.

At the structural level (where the user wishes to manipulate statements and sets of statements as units), NLS permits the user to identify statements by pointing with the mouse; TODAS requires that statements be identified from the keyboard. Considerable flexibility is provided in this operation.

The user may identify a statement directly by typing its statement number or its name; he may also identify it indirectly by specifying its structural relationship to some other statement whose number or name he knows off-hand.

Indirect specification corresponds to the use of NLS commands such as "jump to head," "jump to successor,"

etc., but with the added feature that relationships may be concatenated -- thus the user may, in a single operation, specify a complex relationship such as the successor of the first substatement of the predecessor of a given statement.

A special TODAS capability not yet implemented in NLS is "executable text."

A TODAS statement may consist of the string of characters that a user would type from the keyboard to perform some complex sequence of operations. This statement may then be executed with a special command, and the result will be exactly as if the user had actually typed these characters causing the sequence to be carried out.

The sequence may, in principle, be arbitrarily complex; an executable statement might, for example, contain the following sequence:

- (1) Load a file whose name is specified elsewhere in the current file
- (2) Search this file with the content analyzer, finding statements with a specified pattern of content
- (3) Write these statements out in a temporary "buffer" file
- (4) Reload the original file
- (5) Copy the statements in the "buffer" file into a specified location in the working file.

A special "switch" character may be used in the executable text. When the switch character is encountered, execution of the text is interrupted and control reverts to the keyboard. The user then enters part of the control sequence manually; when he types the switch character from the keyboard, execution of the executable statement resumes at the point where it left off. This feature affords great flexibility, since it allows part of the sequence to be specified ahead of time and part at "execution time."

Besides its primary purpose as a Network user's interface to the NIC, TODAS is used within ARC as a supplemental tool to NLS.

TODAS can be used conveniently for many tasks that do not

require the rapid display response of NLS, and has the advantage of creating significantly less load on the overall timesharing system. We currently have one clerical worker, who is not an NLS user, operating TODAS routinely for entry of information and for some limited retrieval work.

Additionally, we find TODAS useful for remote accessing of our system. We have made TODAS available to selected consultants, who use home terminals with acoustic couplers, and regular ARC personnel occasionally do work from their homes by the same means.

The prototype version of TODAS went into service in September 1969; a second version, with greatly expanded capabilities, became operational early in 1970.

III Output Facility

NLS and TODAS both use the same facilities for producing formatted hard-copy output from NLS/TODAS files.

The devices in ordinary use at ARC for hard-copy output are a line printer that produces upper/lower-case print of adequate quality for local use, and a paper-tape-driven automatic typewriter used for final output of reproducible copy for reports, proposals, etc.

The output-processing program (known as "PASS4") can be controlled by the user to a considerable extent. This is done by means of "directives" embedded in the file text. The directives can be used to reset page parameters, control page numbering, and turn various format features "on" or "off."

For example, directives can be used to suppress indentation of statements or change the amount of indentation, to create "running heads" that are automatically printed at the top of each page, suppress statement numbers, etc. One of the directives causes all directives to be suppressed from the output.

In addition to the line printer and the automatic typewriter, PASS4 can output a file to magnetic tape, appropriately formatted to drive CRT-to-film conversion equipment for production of microfilm.

In all cases, the user may elect to output an entire file or only part of the file. In the latter case, he may cause output to begin at some specified point in the file instead of at the beginning, and he may cause the printout to be limited by the same

kinds of criteria that may be used on the display -- i.e., content analysis, limited number of structural levels, etc.

IV Glossary of Special NLS/TODAS Terminology

BRANCH: A specified statement, plus all of its substructure -- i.e. all of its substatements, plus all of their substatements, etc.

BUG: The mark on the screen which is moved by the mouse and which is used for selecting (pointing to) entities on the display.

When the bug is "active," i.e. when a selection can be made, it appears as an up-arrow; when it is inactive it appears as a plus sign.

CHARACTER: Any letter, digit, punctuation mark, space, tab, or carriage return; an indivisible entity.

CHORD: A combination of keys on the keyset (see KEYSET).

END: The last statement in any branch; specified by specifying the branch.

FILE: A complete tree structure of statements with a single root (the origin statement).

FILENAME: The name of a file. It appears as the first word in the origin statement of an existing file, and must be supplied by the user in creating a new file.

GAP CHARACTER: Any space, tab, or carriage return.

GCHAR: Abbreviation for GAP CHARACTER.

GROUP: A subset of a plex, consisting of all branches from one specified branch to another, inclusive.

HEAD: The first statement in a sublist.

The head is specified by pointing to any statement in the sublist.

INVISIBLE: Any consecutive string of gap characters, bounded by (but not including) printing characters or the end of a statement see PRINTING CHARACTER, GAP CHARACTER, STATEMENT.

Specified by pointing to any character in the string. If a

single printing character lying between two invisibles is pointed to, both invisibles (and the printing character) are selected.

KEYSET: The device at the left-hand side of the console. When a combination of keys (a chord) is depressed on the keyset, the effect is the same as striking a key on the keyboard.

KEYWORD: The name of a "keyword statement."

KEYWORD STATEMENT: A statement which lists, in a special format, the names of all statements in the same file that fall into some arbitrary category.

The "keyword system" of NLS/TODAS commands, operating upon keyword statements, performs information-retrieval operations based on the sets of statements defined in keyword statements.

LABEL: A string of text placed in a picture by means of a command in the vector package.

LEVADJ: The specification of level when a statement, branch, plex, or group is newly created or moved.

LEVEL: The "rank" of a statement (see STATEMENT) in the hierarchy of the file (see FILE).

The level is equal to the number of fields of letters or digits in the statement number; thus Statement 3 is a first-level statement, Statement 4a10g3 is a fifth-level statement, etc. Level is of great importance in understanding the hierarchical structure of an NLS file.

MOUSE: The device at the right-hand side of the keyboard. When it is rolled around on the tabletop, it causes the bug to move correspondingly.

NAME: If the first word of a statement is enclosed in parentheses, it is the NAME of the statement.

The command Jump to Name can then be used to place the statement at the top of the display. This is done by entering the name from the keyboard or keyset, or by finding an occurrence of the name as text on the display and pointing to it with the bug.

ORIGIN: The first statement in a file; it contains information about the file, plus any other text the user inserts. It has a

level of 0, and hence no statement number.

PATTERN: A string of special-language text in a statement which may be compiled via the command Execute Content Analyzer. When compiled, it produces a program that is used by the content-analyzer feature.

PCCHAR: Abbreviation for PRINTING CHARACTER.

PLEX: Another name for a SUBSTRUCTURE, used in command specifications.

A plex is specified by pointing to any one of its highest-level statements.

POINTER: A string of up to three characters which is attached to some character in the text with the Pointer Fix command.

PREDECESSOR: The statement preceding a specified statement in a SUBLIST.

PRINTING CHARACTER: Any letter, digit, or punctuation mark.

SOURCE: The statement of which a specified statement is a substatement.

SIGNATURE: Information stored with a statement (and displayed on command) giving the initials of the user who created the statement (or most recently modified it) and the time and date when this occurred.

STATEMENT: The basic structural unit of a file of text in NLS. Formally, it is a string of text and/or pictures which is bounded at the beginning by the end of the previous statement or the beginning of the file, and bounded at the end by the beginning of another statement or the end of the file.

Statements are arranged in a tree structure or hierarchy and are assigned "statement numbers" which indicate their position in the structure. Each statement has a number, made up of alternating fields of digits and letters; the number of fields indicates the "level" of the statement (see LEVEL).

A statement is specified by pointing to any character in the string.

SUBLIST: The set of all substatements of a specified statement (not including the substatements of the substatements).

SUBSTATEMENT: A statement "X" is called a substatement of another statement "Y" if it is deeper in the structure than "Y," if it follows "Y," and if there is no intervening higher-order statement. "Y" is called the source of "X." The statement number of "X" will be the same as that of "Y" except that it will have one more field at the end. The value of this field gives its ordinal position in a "sublist" of the substatements of "Y."

A substatement is specified by pointing to the source statement.

SUBSTRUCTURE: The set of all substatements of a specified statement, plus all their substatements, etc. until no more are found. The set of all branches defined by statements in the sublist of a given statement.

SUCCESSOR: The statement following a specified statement in a sublist.

TAIL: The last statement in a sublist.

The tail is specified by pointing to any statement in the sublist.

TEXT: Any string of characters within a statement, bounded by (and including) two specified characters: see CHARACTER, STATEMENT.

TRAIL: A set of statements in a file, which can be displayed sequentially by using the trail feature.

VECTOR: A line in a picture.

VISIBLE: Any consecutive string of printing characters, bounded by (but not including) gap characters or the end of a statement: see PRINTING CHARACTER, GAP CHARACTER, STATEMENT.

Specified by pointing to any character in the string. If a single gap character between two visibles is pointed to, then both visibles (and the gap character) are specified.

WORD: Any consecutive string of letters and/or digits, bounded by (but not including) any other types of characters or the end of a statement: see STATEMENT.

Specified by pointing to any character in the string. If a single character is pointed to which is not a letter or digit and lies between two words, then both words (and the single

character) are specified.

Appendix B
THE DIALOGUE SUPPORT SYSTEM (DSS) AND THE JOURNAL

I Preface

For his dissertation study at Stanford University, Dr. David A. Evans (then an ARC staff member and associated with the Management Systems Research Activity) developed the case for augmentation of planning teams.

His thesis (Ref. 1), written with NLS, is over five hundred pages in length. In it he presents for the planning community a broad description of ARC's augmentation approach, developments achieved by ARC, and extrapolations relevant to the planning community.

As a special case study, Dr. Evans integrated the considerations and possibilities for the Dialogue Support System, as developed within the ARC over a number of years and as studied specially by Evans under this contract.

Selected extracts from his thesis, slightly condensed, are included below as a good source of relevant concept material about the DSS. These may be considered as trial design notes; the final designs for the various parts of the DSS, and their order of development, are yet to be developed.

II Basic Components of the Dialogue Support System (DSS)

The DSS can be considered to have two basic parts: (1) the Journal, and (2) a set of NLS features especially designed to operate on the Journal.

A. The Journal

One of the most dramatic things NLS enables its user to do is operate on and maintain extremely "plastic" and malleable records of his thought and work.

This ever-changing plasticity is the root of basic difficulties in extending NLS for dialogue support. When members of a team are contributing to a plan or design, one of the most important things is that the "targets" of their contributions remain stationary, as if in a diary, or journal. Ironically, the design of a "Journal" to maintain stationary-target records of the transactions of members of a team proved to be innovative in the NLS environment, whereas it would be "normal" if we were dealing with simple pencil and paper.

The Journal is a special repository for NLS files which may be "sent to the Journal" and no longer modified, or changed in any way.

The design objective of the Journal is to provide the basis for evolution of a diary for a team, sufficiently rich to play the same role as a personal diary plays when used for record keeping, and as the basis for composition, reflection, and extended memory.

B. Operations Based on Journal Entries

The second component of the DSS is a collection of special NLS features, designed to make the Journal useful as the basis for supporting team dialogue.

The Journal provides the team members with a chronicle of their contributions to plans and designs. NLS, as extended for use as part of the DSS, is a vehicle that (for example) enables team members to annotate contributions from others, to call for specific action, to make synopses of records relevant to specific issues, and to make contributions to the evolution of plans and designs that are efficiently and appropriately integrated and connected to the entire record of activity.

At another level, NLS is a vehicle enabling team members to "browse" in the Journal, to arrive quickly and efficiently at an understanding of the status of plans and designs that are being documented, monitored, or evolved through the medium of the DSS.

Interspersed with this and the previous roles, extended NLS features enable team members to retrieve information from the Journal, to modify and update this information, and to return it to the Journal without destroying the original contributions.

III Design of Architecture for the Journal

A. Introduction

The boundary between the Journal proper and the NLS features that support it is not clearly defined, as those features necessary for servicing the Journal also, indirectly, support the special DSS features. However, the discussion can be simplified by means of this division.

B. Stationary Targets

The ideal record system for dialogue support would be some large, central, evolving record that would keep track of the team's activity as team members contributed modifications, new

ideas, new designs, specifications, and so on, over time. We have only to consider the problems raised by the basic file-handling operations of the current NLS to appreciate the difficulty of creating such an evolving record of transactions.

In any attempt to use files for dialogue purposes, the first problem encountered arises from multiple access to files. When a file is strictly the "property" of its author, dealing with material for which he alone has prime responsibility, the file owner can quite easily keep track of its development.

However, when several individuals make active use of a file, it becomes very difficult for the individuals to avoid canceling each other's work or otherwise interfering with each other. They cannot all access the file simultaneously, and so copies are created; soon there are multiple copies, each copy containing changes and additions made independently by various users. It is then impossible, in the general case, to put these copies back together in such a way that all the work done on the separate copies is preserved.

The problem is much like trying to hit a moving target in the dark, and the desired solution is to find some way to make the target stop moving -- hence the phrase "stationary targets." The existing capabilities of NLS and the file-handling facilities used by NLS are not adequate for achieving this.

For example, it would be possible with existing capabilities to give all files a read-only status, so that once a file was created it could never be modified. This would overcome many of the problems of multiple access; however, it would also destroy most of the power and usefulness of NLS as a tool for manipulating information.

Likewise, it would be possible to give all files a public read/write status, permitting any member of the team to modify any file at will. It can be seen that this would lead to immediate chaos: a team member working on a file and wishing to make reference to another file would have no assurance that the referenced file still contained the same information as when he looked at it last.

The concept of the Journal is a way to create stationary targets without the crippling effect of a blanket read-only policy or the anarchy of a blanket public read/write policy. Files "entered in the Journal" have, in effect, read-only status, but numerous capabilities are added to compensate for

this; moreover, the Journal contains only selected files which are considered to be "ready" to become stationary targets.

C. The Journal

The Journal is a public repository for information of concern to the team of users. A file sent to the Journal becomes a public record. In principle, at least, it cannot in any way be altered, or retracted.

The author has "gone on record" with the statement made by the file's content. He may keep a copy of the file entered in the Journal, and make modifications and corrections in that copy, but cannot replace the original file in the Journal by over-writing it with the revised version. Both the original and revised versions may be entered in the Journal.

A basic Journal function is to provide users with mechanism and aids to recognize that "later versions" in the Journal have been entered, and to provide users with features to enable them to retrieve and display the multiple versions of a given file.

In keeping with other (non-computerized) Journals, the only ordering imposed on Journal entries is chronological.

In NLS, "Journal" becomes a distinct user name, with a status similar to all other users.

However, the Journal adds a second distinct domain of files to the NLS file universe. Journal files have special features. They are all read-only. They possess two parts -- the text/graphics portions written by their author, and blocks of data containing information added to the file after submission to the Journal.

The first component is totally frozen: once a file is "sent to the Journal" the "maximum" user representation for that file may not be subsequently altered.

But the second component, data blocks, may be changed through the addition of new data over time.

1. Journal Entries

Although we have been discussing "files" in the Journal, we should refer to a module of information in the Journal as a

"entry." From the viewpoint of the NLS file system, an entry is synonymous with a file. However, we wish to emphasize the notion of collecting information from many files together into one module, and sending that module to the Journal as an entry. For this reason, we will persist with the terminology "entry" rather than "file" when discussing the Journal from the point of view of a user (contrasted to the viewpoint of the system).

D. Sending an Entry to the Journal

Because of the existence of two file universes (regular NLS files, and Journal entries) a user is not compelled to submit all of his files to public scrutiny.

He may keep his personal collection of files containing his notes, plans, special reminders, etc., separate from the collection of files he submits to the Journal.

Within this personal collection he retains the option of controlling read and write access by other users. He may, for instance, have several files that contain private/confidential information that is of no concern to the team as a whole.

However, the decision to submit one of his own files to the Journal is not totally the prerogative of the user himself, unless all his files have private status.

Files stored under a given user name, with other than private status, may be entered to the Journal by any other user. This is similar to the procedure of having testimony, or a speech, or other data, read into the (Congressional) Record.

However, in most cases, Journal entries are submitted by the user who has the file (or component files) stored under his name, as part of the standard NLS file universe.

For one user to submit another's file to the Journal, he must first load that file, make a temporary copy, and submit that copy as a Journal entry as if it was one of his own "normal" NLS files.

Entering a file to the Journal involves the following operations:

- (1) A copy of the file being submitted is made.

(2) That copy is again copied, by the system, and (automatically) written as a new file under the user name "Journal." It is given a new name, which is a unique "Journal Entry Number," and set to read-only status.

(3) The user submitting this file is given a "receipt" by the system, indicating that entry to the Journal has been successful.

The result is that a "snapshot" of the user's file has been recorded as a Journal entry. The user has complete control over the VIEWSPECS controlling the view and amount of the file submitted to the Journal. For instance, if he so chooses, the user may submit only the first level statements in the file. Or he may submit only selected statements in the file -- for instance, only those that satisfy a specific content pattern. He may, of course, choose to employ no special VIEWSPECS, and submit the entire file to the Journal. The VIEWSPECS used at time of entry to the Journal determine the maximum subsequent view for that Journal entry.

Subsequent readers of the Journal entry may employ all available VIEWSPECS to help them study the content of the entry, but are constrained to this "maximum" view. This means, for example, if a file is submitted to the Journal with a 1-1 VIEWSPEC (i.e., only top level statements, and only one line of these), subsequent readers can view no more information in that entry, other than the 1-1 view, even if he uses a VIEWSPEC such as ALL-ALL (i.e., all statements, and all lines of each statement).

Thus the result of this entry procedure is the creation of a new read-only file, a stationary target, under the user name Journal, with a unique Journal Entry Number as its name.

E. Journal Entry Linkage Systems

Once we have procedures for submitting entries to the Journal, the next major need concerns linking the individual stationary targets -- the Journal entries -- into a fabric of interconnected information.

Interfile links may be used to refer to specific locations in a file from any arbitrary location in another file. The difficulty in this interfile linkage system is that there is no way for a user to discover that a particular entity (e.g., a specific statement) in the file he is reading is being referred to by links embedded in other files, or embedded in other

statements within the same file. This basic weakness leads to indiscriminate deletion or alteration of files.

To solve this problem in the DSS, Journal entries will have "backlinks." This means that when a link is established in a file (for instance, a file not in the Journal), a special marker will be written automatically by NLS in the appropriate location of the referent file, indicating that a link is pointing at that entity.

This marker will give subsequent readers of the referent file a visual signal that the marked entity is the target of a link in another file. A new NLS command, JUMP BACKLINK, will make it possible for the user to jump from the entity in the referent file "back" to the statement containing the link in the source file.

There are five cases of file-pair linkages that produce problems:

- (1) Linkage between two standard NLS files, A and B, from A to B, and file A subsequently becomes a Journal entry.

Problem: The link in A continues to refer to B, and is unaware of the formation of a Journal entry from B. If B is deleted, the link points to a non-existent file.

Need: Additional bookkeeping to redirect links to the appropriate Journal entry if B is deleted or otherwise modified to make the link inappropriate.

- (2) Linkage between two standard NLS files, A and B, from A to B, and B subsequently becomes a Journal entry.

Problem: The backlink attached to the referent entity in B points back to A, and is unaware of the Journal entry made from A at a later date. If A is deleted after its copy is sent to the Journal, subsequent efforts to JUMP BACKLINK on the backlink marker from A in B will yield a "no such" message.

Need: Additional bookkeeping to redirect the backlink to the appropriate Journal entry if A is ever deleted or otherwise modified to make the backlink inappropriate. This leads to the concept of indirect linking.

(3) Linkages between two standard NLS files, A and B, from A to B, and both A and B subsequently become Journal entries.

Combination of problems and needs of Cases 1 and 2.

(4) Linkage from a Journal entry to a standard NLS file that subsequently becomes a Journal entry.

Problem: Link in the Journal entry is unaware of the existence of the Journal entry made from B.

Need: Bookkeeping necessary to redirect the link, if requested, to the appropriate Journal entry if so requested by the user.

(5) Linkage from a standard NLS file to a Journal entry, and the standard NLS file subsequently becomes a Journal entry.

Same as Case 4 except we are concerned with backlinks rather than links.

F. Other Basic Journal Needs

In our first-pass discussion of Journal architecture and needs, we should consider two additional general needs, archiving and cataloguing.

Archiving is necessary because the current system has limited storage area for files accessible to NLS. The only mass storage devices presently available in the ARC facility are magnetic tapes, and so, at first, the Journal will have a sequential archive. All Journal entries have archival copies. The archival system provides a back-up to the colon copy of a Journal entry in case of disaster, and a large tertiary storage area for those entries not frequently referenced, that do not have to be kept continually in colon file storage on the disk.

Major archiving problems arise because of additional data (including backlinks) associated with an entry after it is submitted to the Journal.

Files are allocated a finite number of blocks on a magnetic tape at the time they are written. Data added after the entry is made may be written in this "slop" area until it is filled. But from then on, these data must be stored elsewhere. Only minor problems arise if

the additional data can be stored elsewhere on the same tape, with a link from the original entry to a special file, elsewhere on that tape, associated with that entry, containing additional data.

However, when the tape is filled, these data have to be stored on a separate tape. This causes considerable difficulty when retrieving the entry and its associated data from the archive. There is no simple solution to this problem while magnetic tape is the archival media. These problems will not arise with random-access mass-storage media.

The final basic Journal feature is a catalogue. Obviously, a Journal reader requires a guide to the contents of the Journal, and this is provided by the catalogue.

The Journal Catalogue will have three principal parts:

- (1) Subject index
- (2) Citation list for Journal entries
- (3) Keyword lists.

IV Design for Detailed NLS Features to Support DSS

A. Submission of an Entry to the Journal

1. Entry/Receipt Procedure

When a file is submitted to the Journal, the first operations are concerned with creating a new Journal entry, allocating a unique number to that entry, and giving the sender a receipt. This receipt acknowledges the entry has been made successfully, and supplies the sender with sufficient information to enable him to locate and retrieve the entry at a later date. Details of this procedure are illustrated in the following scenario.

a. Scenario: Entry/Receipt Procedure

- (1) Assume the user, X, has assembled a file (X,X1) to be submitted to the Journal.
- (2) He activates the new NLS command "ENTER FILE TO JOURNAL filename," entering the filename X1, as the operand for this command.

(3) NLS makes a copy of the file (X,X1) as a temporary file, (JOURNAL,T1), i.e., under the user name "Journal."

(4) Immediately after making this new file, the system checks a special record, containing a "Journal Entry Number," taking note of the time and date this check is made. Journal Entry Numbers have the form "NNNJMMY."

"NNN" is a serial number, in the range 1 to z where z is arbitrarily large.

"J" is the literal character "J," indicating that the number refers to a Journal entry.

"MM" is the month the entry was submitted (e.g., 11 = November).

"Y" is the year the entry was submitted (e.g., 9 = 1969).

The serial numbers, NNN, are initialized at the start of each month.

Example: If 4562J119 is the last entry submitted to the Journal in the month of November, 1969 (indicating that 4562 entries were submitted in that month), the next Journal entry would be allocated the number 1J129.

Assume that the number in this location at the time of this particular access was 457J119, and the exact time of access was 1451:30, on 11/13/69. Once this number has been secured, the system updates the latest Journal Entry Number in this location (to 457+1 = 458).

The system now copies the file (JOURNAL,T1) to a new file -- a Journal entry with file name 457J119. It sets the status of this file to public read-only, and notes the time and date of completion of making this Journal entry: 1451:45, 11/13/69.

Once this Journal entry has been made, the system returns a message "FILE (X,X1) ENTERED TO JOURNAL AS NUMBER 457J119 AT 1457:45" to the sender (user X).

This message remains on user X's display until a command accept (CA) is entered. Entering the CA releases the file (X,X1) for normal operations, and

redisplay the file. User X is now free to continue his normal work.

2. Data Assembly Procedures at Input Time

The time an entry is submitted to the Journal is an opportune time to capture data associated with the entry. The Journal entry procedure will contain additional operations, in which the system interrogates the user to obtain an abstract and special descriptor tags for the entry. The abstract will be used in the Journal catalogue. Descriptor tags will be used for retrieval of entries.

3. Collection System

Part of the Journal entry system gives the user special aids for assembling the entry before actual submission. These are compound operations, combining several simpler ones. These simpler operations include file merging and the "executable statement" capability.

B. Linkages

Special linking features will be added to NLS to support the DSS needs. One of the most important classes of these new features concerns NLS links.

1. "Link" as an NLS Entity

In the current NLS a link is a simple text construct; it is not a special entity, in the way that characters, words, and statements (for instance) are entities.

There is no command DELETE LINK in current NLS. A link may be deleted using the normal DELETE TEXT command, requiring two bug selections, one at each of the link parentheses.

A special NLS entity "link" will be added to NLS. This will be of particular importance in combination with indirect linking and executable statement operations.

To insert a link, the new command INSERT LINK is used. This command requests user input of data necessary to construct the link, and organizes these data in the appropriate syntax (see below).

2. New NLS Link Syntax

a. Additional Link Data

Additional data will be added to the current NLS link construct. These data are (a) link type, (b) time and date the link was first constructed, or last "stamped," and (c) improved resolution to identify link referents.

Link type data are one or more descriptors, being a simple text name, or collection of names, indicating membership of a class, or classes.

Example: Possible link types would be "footnote," "comment," "rebuttal," "owner-evans," etc. A link "owner" could be different from the owner of the file in which the link resided. The definition of these types and their respective mnemonics would be determined by agreement among DSS users.

A most important addition to NLS links will be the added power to refer to ANY entity. In the current version of NLS, a link may point only to statement entities.

With greater resolution for link references, for instance, a link may be constructed to refer specifically to another link. This is the basis for indirect linking, to be discussed below.

b. Possible Syntax for New NLS Link Entity

<TYPE;DATE,TIME> (USERNAME, FILENAME,
LOCENTITY:VIEWSPECS)

TYPE is any number of descriptor mnemonics defining the type of the link. Each descriptor would be delimited by a comma.

MMDDYY HHHH:SS is the date and time the link was created, or the date and time the link was last "stamped," in the format <month, day, year, hour, second>.

At any time, the link's owner may initialize the time and date for the link, using a date-time "stamping" command.

USERNAME, FILENAME, and VIEWSPEC have the same meaning as in current NLS links.

LOCENTITY identifies a specific target entity. Detailed syntax for the LOCENTITY may be arbitrarily complex. The following example indicates a simple statement-number syntax.

c. Example

```
<comm,urg,Evans;09/17/69 0014:44>  
(Engelbart,plans,m-P:xi)
```

TYPE is "comm,urg,Evans"

DATE,TIME is "09/17/69 0014:44"

USERNAME is "Engelbart"

FILENAME is "plans"

LOCENTITY is "m-P" (the marker "P")

VIEWSPECS are xi, meaning display only one line of top-level statements, and switch on the content analyzer.

This link refers to the entity with marker "P" affixed ("m-P") in the file ":plans" owned by user name "Engelbart." It points from a comment ("comm") that is urgent ("urg"), and should be brought to the attention of user name "Evans." The link was last stamped 09/17/69 at 0014:44.

3. New VIEWSPECS for Links

Increased link complexity demands more powerful VIEWSPECS to simplify displaying the link construct, so links do not make the remainder of the text illegible.

Additional VIEWSPECS will be available for totally or partially suppressing display of the link construct. For instance, the user could control which fields in the link were displayed at the link's location in a statement (this VIEWSPEC would apply to the entire display). If the link was to be totally suppressed, an additional VIEWSPEC would allow the user to control whether or not special "link markers" were displayed at the link's normal location.

A user would interrogate an individual link marker, to display the particular link represented by that marker,

without displaying all links.

4. Links Not Embedded Directly in Text

Because of the "stationary target" concept and the frequent need to attach links to existing Journal entries, it will be necessary to have a new NLS command to enable a user to associate an NLS link with any selected text entity, but have that link displayed only as an overlay to the file, rather than an integral part of the normal text. Link markers, similar to those used for backlinking, will be used to indicate the presence of one of these links. New NLS commands will be available to enable the user to control the display of the link and markers.

5. Indirect Linking

Once it is possible to "aim" a link at any arbitrary entity, such as another link, or at a simple character in a statement, indirect linking becomes possible. The following example illustrates detailed operation for indirect linking.

Example: The following link is displayed in a statement of the file (Evans,ddd): <comm;>(Engelbart,plans,m-P:). Note that the date-time field has been suppressed by the new link VIEWSPECS described previously. This link is embedded in a statement (or branch) constituting a comment on its DIRECT target.

In the file (Engelbart,plans) there is a marker "P" affixed to a character just preceding another link, as follows: <P>xx yy cc <comm;>(Evans,rrr,l2b:w). This link is a comment on l2b in the file (Evans,:rrr).

Use of the new command JUMP INDIRECT LINK, with the original link as operand, causes the statement l2b to be displayed under the control of VIEWSPEC "w" (all lines of all statements).

6. Backlinks

The most important additions to existing NLS linking features for use in the DSS are the backlink operations.

Backlinking means that a special executable link marker is deposited in the referent being pointed at by a link. This enables a user, viewing the referent entity, to "JUMP BACKLINK" and display the entity containing the original

link.

The existence of an NLS link reference to any displayed NLS entity will be indicated by special backlink markers. Display of these markers will be under user control in a manner similar to link markers, described previously.

A user may interrogate a backlink marker, to have data on the source entity displayed. Execution of the new command JUMP BACKLINK with a backlink marker as operand displays the source entity at the top of the display.

Indirect backlinking will also be available. Indirect backlink jumping means that a user executes JUMP BACKLINK INDIRECT, and the system displays the statement containing the link that points at the source of the backlink marker entered as the operand for this command.

7. Remote Linking

The basic concept for remote linking is that of attaching the "head" of a link to its referent entity, followed by insertion of the link itself in the source entity, remote from the referent, at some later time.

This may be accomplished by the following steps:

- (1) Assigning a temporary marker to yet another entity, "link referent"
- (2) Depositing that marker at the appropriate location in the referent statement
- (3) Later, while inserting the basic link construct in the source statement, calling for the referent entity data to be inserted in the link by using a special INSERT REFERENT DATA command, entering the referent marker as operand.

This type of operation depends upon each user having at least two NLS files open simultaneously. If links and backlinks are considered to be completely symmetrical, this procedure may be used interchangeably with the conventional INSERT LINK command.

C. Copying a Journal Entry

A problem arises when a Journal entry, stored as a colon file, is copied to a new filename. All backlink markers are retained, but the links generating these markers continue to refer to the original Journal entry, and do not point at the new file. Thus an additional type of backlink is produced -- one that has no forward-pointing link associated with it.

These asymmetrical backlink markers make it possible to jump to files and entries that referred to the original entry. They may be deleted if judged to be inappropriate for the new file.

At the time the new file is created, the system will automatically insert a link in the file's header statement, pointing at the header statement in the Journal entry from which it has been copied, and depositing a backlink marker in the header of the Journal entry.

D. Ordered Sets

A set is a special new NLS entity -- it is a collection of other entities (e.g., of characters, files, statements, links, other sets, etc.). The design and implementation of operation associated with sets is a complex problem. The following indicates what seem to be the most promising possibilities.

An "ordered" set has a specified order associated with its member entities. Sets are given unique names for identification. For convenience, a set will be attached to a "parent" file, selected arbitrarily by the user. (Evans,XXX) is the set named "XXX" owned by the user name "Evans." Set names are similar to statement names, except they must be unique over the entire universe of a user's files -- it is not possible to have a set named "XXX" associated with the file :ccc and another set "XXX" associated with the file :ddd, if both :ccc and :ddd are owned by the same user. However, different users may own sets with the same name.

1. Admission to a Set

Other NLS entities, including other sets, may be "admitted" to a set, using the command "ADMIT <entity> TO SET <setname>", and entering the appropriate operands.

"Entity" is the NLS entity selected or specified by the user; "setname" is the name of an existing set -- the set

to which the entity is to be admitted.

Not only entities, but specific views and specific subsets of entities, may be admitted to a set.

Example: The first line of the first two levels of statements in a file satisfying a given content pattern, may be admitted to a set. The remainder of that file, unless specifically admitted on another occasion, does not belong to the set.

2. Direct and Indirect Use of Sets

There are three modes for using sets: "normal," "direct," and "indirect."

"Normal" mode corresponds to normal NLS usage in which the set entity has the same status as normal NLS entities (words, characters, etc.).

Thus in normal mode, the command DELETE SET erases the set whose name is given as an operand. Note that the set is erased, not the members of the set.

In "direct" mode, operations performed on a set produce changes in the actual entities admitted to the set.

Example: A (hypothetical) command "DELETE WORD m-spec IN SET (evans,X)" is entered; "spec" is an NLS marker name. Upon execution, in direct mode, all words so marked in the entities that are members of the set (evans,X) will actually be deleted. That is, they will be deleted in the same sense as if the user displayed each entity in the set containing the marker, and manually deleted the marked word, followed by the command OUTPUT FILE.

Entities changed through operations performed on sets in "direct" mode remain changed after the system is returned to "normal" mode.

In "indirect" mode, operations performed on entities that are members of a set (by using the set name itself as the operand) produce changes in those entities ONLY while the user views them "through" the set.

For instance, if in the previous example the same operation was performed in "indirect" mode, the marked words would not be deleted in the files containing the

marked entities in question, but would only "appear" to be deleted when the viewer was working with the set [evans,X] controlling the entities he could display. This appearance would be negated as soon as the user returned to display any member-file in normal mode.

3. Open and Closed Sets

a. Closed Sets

A closed set is one whose membership is specified explicitly, i.e., there is a finite fully determined membership list associated with the set. For example, statement entities might be specified by a list of NLS links. There are three types of closed sets: frozen, unfrozen, and mixed.

A frozen closed set retains the exact content and structure of each entity, in the state in which it was originally admitted to the set. Even if (say) a member statement is deleted, a "copy" is retained in the set.

An unfrozen closed set retains a finite membership, but permits each member entity to adopt its latest actual state. For example, a whole file, containing three statements admitted to an unfrozen closed set on day 1, subsequently undergoes major modifications. If the set is used as an operand on day 3 (after the modifications), the file's state at that time is used.

A mixed set contains entities whose frozen/unfrozen status is determined individually. In other words, a set may contain some entities whose original status is retained, and some whose status is the latest status of the entity itself.

b. Open Sets

An open set is one whose membership is not fixed by explicit identification of its member entities, but rather by the specification of conditions to be met to admit member entities.

For example, an open set's membership may be determined by those statements in a given file universe that satisfy a given content pattern.

On day 1, this may yield a different membership than on day 4, if modifications were made to files in that universe during this period.

4. Set Operations

There are two major and distinct classes of operations associated with sets -- operations on sets, and operations within sets. The distinctions between these classes are important.

a. Operations on Sets

Operations on sets use entire sets as operands.

Simple Operations on Sets

These operations include the standard NLS operands -- INSERT, DELETE, REPLACE, etc., in addition to a new class of commands -- set-theoretic operations.

INSERT SET creates a new set.

REPLACE SET makes it possible for a user to make a new set as the union of one or more existing sets, and to simultaneously delete the original sets (their names, not members).

DELETE SET erases the set (but not its members).

Set-Theoretic Operations on Sets

There will be new NLS commands to enable a user to perform set-theoretic operations on sets. The following set-theoretic commands will be available: UNION, INTERSECTION, COMPLEMENT, and DIFFERENCE, where each operation has its usual mathematical meaning.

b. Operations Within Sets

Operations within sets have entirely different meanings from operations on sets, and from operations on member entities outside the influence of the set construct.

When under the control of operations within sets, the conventional NLS commands take on the following meaning:

MOVE: Change the ORDER of member entities in the set.

DELETE: Remove the operand-entity from membership of the set.

COPY: Include the operand-entity once more in the set membership (in a different position within the set's order).

INSERT: Admit the operand-entity to membership in the set.

REPLACE: Replace the member entity selected as operand with the entity selected. The entity selected as a replacement may or may not be a member of the set.

E. Executable Statements

An executable statement will be a new text construct, using the current NLS statement as a basis. NLS commands will be pre-specified as a text string in an executable statement. They will be executed by using the command EXECUTE STATEMENT, giving the statement number of the statement as operand.

An executable statement will be the means to effect compound or concatenated operations, including set operations. The structure and meaning of the executable statement features can best be illustrated by examples.

Example: The following is an executable statement.

```
(XXX) (evans,sss,12:x) (Engelbart,plans,2:w) E C CA  
["retrieve "] OR ["Retrieve"] ; CA (evans,rrr,:w1) END
```

(1) By activating the command EXECUTE STATEMENT, and entering the operand "XXX" (the name of the executable statement), followed by a single CA, the first link will be executed as if JUMP FILE LINK was used with that link as its operand.

(2) The user views the file (evans,sss) with statement 12 at the top of the screen, displaying only the first lines of subsequent top-level statements in the file.

(3) A second CA causes the second link to be executed.

(4) The user views the file (engelbart,plans), with statement 2 at the top of the screen, displaying all lines of all statements.

(5) A third CA causes the content pattern ["retrieve/ OR ["Retrieve"] to be compiled, automatically followed by the execution of the last link. Note that the VIEWSPEC "i" in the last link activates the pattern.

(6) The result is that the file (Evans,rrr) is searched; all statements containing the text construct "retrieve" or "Retrieve" are displayed.

Example: The following executable statement illustrates more complex operations on sets.

```
(YYY) [DOD] = [ARMY] UNION [NAVY] ; [USA] = [DOD]  
INTERSECTION [MIC] ;E C CA ["weapon"] ; CA  
(Nixon,[USA],:w1) CA DISPLAY:w OUTPUT FILE ':arsenal'  
DELETE SET [DOD] AND SET [USA] END
```

(1) The command EXECUTE STATEMENT is executed with the operand YYY, the name of the statement.

(2) A CA causes a new set "DOD" to be formed as the union of the two existing sets "army and "navy." This set will be attached to the file containing the executable statement.

(3) Another CA causes a second set, "USA" to be formed as the intersection of the two sets "DOD" and "MIC."

(5) Another CA causes the content pattern "weapon" to be compiled, immediately followed by execution of the link transferring control to the first entity containing the text construct "weapon" in the set "USA" (which is owned by the user "Nixon").

(5) The system searches all entities in this set, and displays, under VIEWSPEC control "w" (all lines of all statements) those statements containing the text string "weapon".

(6) A final CA causes this collection of entities to be output as the new file ':arsenal.' Another CA causes both the sets (as distinct from the set membership) [USA] and [DOD] to be deleted.

Example: The following executable statement illustrates how the member entities of a set may be displayed.

```
(ZZZ) DISPLAY:w [HEREANDNOW] END
```

By giving the command EXECUTE STATEMENT with ZZZ as the operand, followed by a CA, all entities in the set "HEREANDNOW" will be displayed, under VIEWSPEC control "w" (all lines of all statements).

Example: The following is an example of simple "chain generation" using an executable statement.

```
(AAA) MARKER=A1 CHAIN (evans,ss,12:gw) (evans,ss,5:gw  
(Engelbart,plans,5:wh) END
```

By giving the command EXECUTE STATEMENT with the operand "AAA", followed by a CA, the display starts with an all-all view of the branch starting with statement 12 in (Evans,:ss). Normal text operations may be performed on this branch. If a second marker A1 is entered, the all-all view of the branch starting with statement 5 in (evans,:ss) is displayed, and so on.

Here a marker is used as the means to advance the view along the chain. This permits normal text operations (requiring CA's) to be performed at each view along the chain.

In all examples, the maximum VIEWSPEC operative on any entity is controlled by the VIEWSPEC assigned to the set member entity itself at the time it was admitted to the set.

F. Entry Descriptors

Descriptors will be attached directly to Journal entries, either at time of entry to the Journal, or at some later date. These descriptors will cover at least the following classes:

(1) Subject matter/type of entry

Examples: comment; message; announcement; injunction

(2) Urgency

Examples: urgent; not urgent

(3) Names of users whose attention is sought

Example: attention: evans, engelbart.

(4) Author/source of entry

Example: author: evans;

(5) Date/time of entry to Journal

Example: entered 9/26/69 1006:30

G. Interrogation

Commands will be available to enable a user to interrogate a Journal entry in order to ask the following types of questions:

(a) Which Journal entries or other files are pointing at the interrogated entry?

(b) To which sets does the interrogated entry belong?

When interrogating to determine which entries or other files are pointing at the entry, the user will be able to control the universe over which the search for these entries is to be performed.

For instance, the user may ask for only those entries that point at the interrogated entry, or are attached by links of a specified type, from entries of another specified type, that were made after a specified date.

Example: Display Journal entries of type "comment" or "injunction" that are attached with link types "urgent" made after 8/12/69 to Journal entry Number XXXXX.

Example: Display those members of the set [evans,XXX] admitted to the set after 10/4/69.

H. Miscellaneous New NLS Features

Numerous new NLS features will have a major effect on the usefulness of the DSS, although they are not designed exclusively for DSS usage. These features include split screens, file merging, new VIEWSPECS, and "file history."

1. Split Screen

The "split screen" feature generalizes the characteristics of the "freezing" option in the current version of NLS. With a split screen, the user is able to display two different views of the same file, or two different and independent views of any two files, one on each side of the screen. He will be able to work with the displayed

information in each "window" as if it was a separate and independent file. The success of this option depends upon having more than one file open for a given user at any given time. The split screen will make interfile editing, and more complex file merging, easy and useful.

2. File Merging

The split screen and other new features make the capability for merging any two files to form a third composite file a necessity. In the current version of NLS, only the simplest file merging operation -- appending -- is possible. More useful file merging would include the facility to interleave statements in a specified order, and to transfer pictures from one file to another.

3. File History

Keeping track of a file's history becomes more important in the Journal and DSS than in current NLS operations. For this reason a new NLS feature will be added to capture all necessary identification information from the source file every time a file is output or copied. This information may be copied directly from the header statement of the source file, and written into the header statement of the object file at the time it is created.

Example: The following is an example of a standard file header.

```
:XVIII, 9/26/69 1209:30 DAE;
```

Here :XVIII is the filename; 9/26/69 1209:30 is the date and time the file was last output to the name :XVIII, and DAE are the initials of the file owner.

Suppose the file :XVIII is output to the new file name ":CHAP18".

After the operation is completed, the header of the object file (:CHAP18) reads as follows:

```
:CHAP18, 9/26/69 1211:45 DAE; (evans,XVIII,:)  
9/26/69 1211:45;
```

The system has rewritten the source file's header data as an NLS link following the object file's conventional header data. Note that as later versions

of :CHAP18 are made, data preceding the first semicolon changes. With subsequent copy operations, or output file operations to new filenames, these data from the file :XVIII will be retained in the new file's header, along with all records of subsequent operations.

I. Cataloguing

A catalogue of all entries in the Journal will be maintained, providing the main conventional aid for retrieval of these files. The catalogue will have three main sections: a subject index, a keyword list, and citations for Journal entries.

The subject index contains a hierarchical structure of the subjects describing Journal entries, with their respective keywords attached. A user may scan this index and select keywords attached to the subjects that meet his needs.

The Keyword List will contain keywords (as used in the subject index), followed by links pointing at appropriate citations.

The citation for each Journal entry is stored in the catalogue by order of Journal Entry Number. Each citation will constitute an NLS branch, with the Journal Entry Number, and link to the cited Journal entry, as the first-level statement of each branch.

Each such citation branch will contain the entry number, the source filename, the name of the user submitting the entry, the date and time when the entry was submitted, and a list of descriptors for entry.

These data will be stored in a manner that makes them useful for further NLS operations. For example, the data on source filename is stored in the form of a conventional NLS link referring to the source file. Similarly, each catalogue entry contains a link to the Journal entry itself.

1. Retrieval System Based on the Journal Catalogue

The existing NLS keyword retrieval system will be extended for use as the basic retrieval tool for operations on the catalogue. The major drawback of the current system is that lists of citations can be assembled only from within a single file.

For the DSS, this system will be modified to operate across an arbitrary number of files. Such operations, of course, depend upon other features discussed previously (e.g., file merging, the capability of having more than one file open at any instant, etc.).

The standard keyword statement, which currently uses statement names as keyword arguments, will be changed to use full NLS links as keyword arguments.

Example:

```
(key3) This is keyword three * (JOURNAL,135J99,:)  
(Journal,146J99,:)
```

The user will then have the following options:

(1) Assemble the citations derived from a selection of keywords from one or more files (which may themselves be stored in several catalogue files), as a list in one file, and use the standard JUMP LINK command to view the actual Journal entries cited, one by one.

(2) Ask for consecutive display of the actual Journal entries cited, under the control of the VIEWSPECS in the keyword referent links. Consecutive entries cited would be displayed as if part of the same file.

This operation could be accomplished by special new NLS machinery, or by combining the capabilities of executable statements and indirect linking.

In all cases, all current NLS keyword options, including the allocation of weights to keywords, will be available.

Appendix C
REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT

I Introduction

This appendix is an addendum to the previous Hardware Reference Manual, Appendix B of Ref. 3. It consists of a programmer's reference manual for the following equipment:

A line printer (replacing the line-printer description contained in the previous manual)

An inter-core controller for transfers between 940 core and external core ("Xcore")

A Network interface connecting the 940 to the ARPA Network via the Interface Message Processor (IMP)

A precision clock.

II Line Printer

A. General Information

The printer is a Data Products Model M600-11A with 96 characters and a printing speed of about 340 lines per minute. It will accomodate paper from 2-1/2 to 18-1/2 inches in width. Character spacing is 10 per inch and line spacing is 6 per inch. The maximum number of characters per line is 132.

The printer is controlled by EOM instructions and a "unit reference cell" (URC). The URC points to a print buffer resident in core that contains data and control codes. An SKS instruction indicates "printer ready" and an interrupt indicates "end of operation," either normal or error. Error conditions are detected by the controller and an error code written in the URC.

The cells immediately following the URC in core are called "URC+1," "URC+2," etc.

Fixed core assignments for the printer are:

URC	10
Interrupt	211.

B. EOM and SKS Codes

The EOM codes are:

20230106	Initiate
20230406	Reset.

Appendix C

REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT

The "initiate" EOM starts the printer with the word and character designated by the contents of the URC at the time the EOM is given.

The printer controller continues to process the printer buffer until an illegal character or end-of-buffer code is read, or until a "reset" EOM is issued.

An "initiate" EOM given while the printer is busy is ignored.

The "reset" EOM immediately terminates all printing and returns the system to a reset state.

A "reset" EOM given while the printer is disconnected is ignored.

One SKS code is provided for the printer. The code is

04030106 Skip on ready.

This SKS skips if the printer is ready to begin operation. If the printer is not ready, an interrupt is issued when it is made ready.

C. Unit Reference Cell

The URC associated with the printer system has the following format:

0	3	5	8	23

:	:	:	:	:

error			address	

Bits 6-23 contain the absolute address of the first character of the line to be printed (or currently being printed).

Bits 8-23 denote the absolute word address.

Bits 6-7 indicate the character in the word.

A 00 code is the leftmost character. The 11 code is not used but is interpreted as the leftmost character

After a line has been successfully printed, the address

in the URC is updated to point to the first character of the next line.

Bits 0-3 are written by the controller with an error code when errors are detected. Error conditions and codes are described below.

Bits 4-5 are ignored by the controller.

D. Print Buffer

The print buffer is a contiguous sequence of words in core that is interpreted by the printer controller as three 8-bit characters per word.

Characters in the print buffer may be either data characters or control characters.

The control characters are:

373	(NOP) No operation
375	(EOB) End of print buffer
376	(EOL) End of line
377	(NOP) No operation
015	Shift to lower case and lock
035	Shift to lower case for one character
055	Shift to upper case and lock.

An EOL or EOB code causes the current line to be printed with any characters already in the line left-justified.

An EOB code generates an interrupt to the computer after the line is printed and any spacing action has been completed.

The three case-shift codes are self-explanatory. They can appear anywhere within a line of data characters and cause the indicated case-shift actions.

In addition to the explicit control characters, the first character in each line is interpreted as a paper-feed code. These codes are as follows (the word "space" here refers to line spacing, not the "space" character):

020	Space 1 line
021	Space 1 line
022	Space 2 lines
023	Space 3 lines

Appendix C
REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT

024	Space 4 lines
025	Space 5 lines
026	Space 6 lines
027	Space 7 lines
000	Space on channel 0 of format tape
001	Space on channel 1 of format tape
002	Space on channel 2 of format tape
003	Space on channel 3 of format tape
004	Space on channel 4 of format tape
005	Space on channel 5 of format tape
006	Space on channel 6 of format tape
007	Space on channel 7 of format tape.

The action indicated by the space code takes place before the line is printed.

Two successive spacing operations can be caused by sending one of the above space codes followed by "end of line" (376), then another space code.

If no spacing is desired, as when printing the top line on a page, a no-op code (377) should be sent in the first position of that line.

Channel 1 of the format tape is used for "top of form." The number of lines on a page is normally set to 60.

Except for the first character, the print buffer contains only printing characters (including space characters) and control characters. Any other character codes in the print buffer are considered illegal and cause an error condition.

Print buffers may be as large as desired, but no relocation mapping is provided. If a buffer is to extend across a page boundary, the software system must ensure that the two pages are consecutive in memory.

E. Error Conditions

On the detection of any error, an interrupt is issued and the error code is written in the URC.

The error codes and conditions detected are:

000	No error
101	Illegal character code
110	Printer not ready
111	Excessive time.

Zeros in the error-code bits of the URC after an interrupt indicate a normal interrupt (printer made ready or EOB).

The 101 code indicates that an illegal character has been detected in the print buffer.

The 110 code indicates printer off-line, paper out, or ribbon failure.

The 111 code indicates that in a normal printing operation, excessive time has been required for printing a line.

The timer is normally set for 2.5 seconds. This error indicates printer failures not detected by other printer error circuits.

F. Character Codes

The printer character codes are given below. The case printed is determined by the shift-control character.

CODE	UPPER	LOWER	CODE	UPPER	LOWER
000	0		040	-	underbar
001	1		041	J	j
002	2		042	K	k
003	3		043	L	l
004	4		044	M	m
005	5		045	N	n
006	6		046	O	o
007	7		047	P	p
010	8		050	Q	q
011	9		051	R	r
012	null		052		
013	*		053	\$	
014	'		054	*	+
015	null		055	null	
016	>		056	;	:
017	null		057		
020	space		060	null	
021	A	a	061	/	%
022	B	b	062	S	s
023	C	c	063	T	t
024	D	d	064	U	u
025	E	e	065	V	v
026	F	f	066	W	w
027	G	g	067	X	x
030	H	h	070	Y	y
031	I	i	071	Z	z
032			072		
033	.		073	,	@
034	)	/	074	(	/
035	null		075		&
036	<	+	076	\	"
037	?	#	077		overbar

III Inter-Core Controller

A. General

The inter-core controller controls transfer of data between external core (often referred to as "Xcore") and 940 core. It has two modes of operation:

(1) A block transfer mode allows the transfer of blocks of up to 2048 words between any two locations in the two cores. This transfer can be between two locations in the same core.

(2) A short transfer mode allows the transfer of short, fixed-length buffers between fixed locations in 940 core and external core.

Fixed core assignments for the inter-core controller are:

URC, 940 core	53
Fixed transfer address, Xcore	100
Interrupt	215.

B. EOM Instructions

Four EOM instructions are used for the inter-core controller.

The EOM codes are:

20230103	Block transfer
20230203	Xcore to 940 fixed transfer
20230303	940 to Xcore fixed transfer
20230403	Disconnect

The EOM actions are:

Block Transfer -- This EOM starts a variable-length transfer. The number of words to be transferred and the starting addresses in source core and destination core are determined by the contents of three consecutive 940 memory cells starting with the URC. Source and destination may be in the same core.

Xcore to 940 fixed transfer -- This EOM initiates a transfer of a fixed number of words beginning at a fixed address in Xcore to a location beginning at the URC in 940 core, starting with the URC address in the 940 computer to a fixed starting address in the external core.

The number of words is determined by a card in the controller and may be set to any number between 1 and 7. The number currently used is 3.

940 to Xcore fixed transfer -- This EOM initiates a transfer of a fixed number of words (same number as above) from 940 core to Xcore, with the same fixed locations in each.

Disconnect -- This EOM terminates any transfer in progress and places the controller in the disconnect state.

C. Unit Reference Cell

The URC and the next two cells have the following coding when used to control a block transfer operation:

0	3	5	8	23

:0	0	0	1:	: : : :

ID	I	word count		

Bits 0-3 contain an identification code. If any other code is detected, the controller disconnects and writes an error code in the URC.

Bit 5 is set to 1 if an interrupt is desired at the completion of the transfer cycle.

Bits 8-23 indicate the number of words to be transferred.

Bits 4 and 6-7 are ignored.

The cell URC+1 contains information relating to the destination of the transfer. It has the following format:

0	3	5	6	23

:0	0	0	1:	: : :

ID	d	destination address		

Bits 0-3 contain an identification code as above.

Bit 5 specifies the destination core. A 1 indicates transfer to 940 core and a 0 indicates transfer to Xcore.

Bits 6-23 designate the first address in the destination core.

The cell URC+2 contains information relating to the source for the transfer. It has the following format:

```

0      3      5 6                                23
-----
:0 0 0 1: : : :                                :
-----
check      D      source address

```

Bits 0-3 contain an identification code as above.

Bit 5 specifies the source core. A 1 indicates transfer from the 940 core and a 0 indicates transfer from Xcore.

Bits 6-23 designate the first address in the source core.

D. Exit Routine

At the end of any transfer, or when an error is detected, the exit routine is performed. This routine writes the URC and then places the unit in its "disconnect" state. The URC is written with the following format:

```

0      2 3      7                                23
-----
:      :0 0 0 0 0:                                :
-----
error      word count

```

Bits 0-2 contain an error code. The errors are reported as follows:

Bit 0 is set to 1 if any error is detected.

Bit 1 is set to 1 for an error in any of the URC locations (incorrect ID code detected).

Bit 2 is set to 1 if the controller waited more than 1 millisecond to gain access to the external core.

Bits 3-7 are set to 0.

Bits 8-23 contain the contents of the word-count register at the end of the transfer. For a successful transfer this will be 0.

An interrupt is issued at the end of the exit routine if called for by the URC, or if any error has been detected. No interrupt is issued for the short transfers.

IV Network Interface

A. General

The network interface provides communication between the 940 and an Interface Message Processor (IMP) on the ARPA Computer Network. The interface operates from message buffers in 940 core. A "linked-buffer" scheme permits flexible memory allocation.

The interface contains two independent logic systems, the input controller and the output controller. The former receives information from the Network, and the latter sends information to the Network.

As seen by the programmer, these two units are almost identical in all aspects except the direction of data flow. Differences between the two are noted in following sections.

The two channels are independent in action, except that they share the same channel into memory. Thus they cannot make simultaneous core accesses.

Fixed locations assigned to the Network interface are:

Receive URC	60
Send URC	70
Receive interrupt	212
Send interrupt	213.

B. Communications with the IMP

Data moving between the Host and the IMP is in the form of serial bit strings with a maximum length of 8096 bits and a maximum rate of one million bits per second.

Details of the communications protocol between the interface and the IMP are covered in Ref. 2.

C. EOM Instructions

EOM Codes are:

20230104 Host up

20230204 Initiate receive
20230304 Initiate send
20230404 Reset.

The "host-up" EOM resets the "host-up timer." This is a timer in the interface controlling a signal to the IMP indicating that the host computer is up. If the timer is not reset at least once a second, indication is given to the IMP that the host is down.

The "initiate receive" EOM enables a "receive" operation. Subsequent to this EOM, data received from the IMP will be written in the "receive" buffers. The EOM must be given for each message received. The controller may be left in the "receive enabled" state indefinitely, waiting for a message from the IMP.

The "initiate send" EOM initiates a "send" operation. Data contained in the "send" buffers will be immediately transmitted to the IMP. A "send" EOM must be given for each message to be transmitted.

The "reset" EOM causes both the controllers to immediately abort any operation in progress and go to the "reset" state.

D. Linked Buffers

Linked buffers are used for both "send" and "receive" messages. The format of the linked buffer is as follows:

The first word of the buffer contains the byte count for the buffer.

If the byte count is zero, the controller goes directly to the next buffer.

A block of n bytes to be transmitted will occupy the $n/3$ core addresses immediately following the byte count, since there are three 8-bit bytes in each 24-bit 940 word. When the last byte does not fall on a 940 word boundary, the action depends on the operation:

In a "send" operation, bytes remaining in the last word are ignored.

In a "receive" operation, bytes remaining in the last word are filled with 0's by the controller.

The last word of the buffer contains the absolute address of the next buffer.

If the last word contains all 0's in the address field, no more buffers are processed and the operation is terminated.

The first buffer of a "send" or "receive" message always begins 2 words after the "send" or "receive" URC, respectively (there are two URCs -- see below).

The maximum message length as determined by the IMP is 8096 bits.

E. The Unit Reference Cells

There are two URC locations for the interface, one for "send" and one for "receive." There are two words at each location, followed by the first message buffer (see above). The URCs have the following format:

First Words:

```
0 1 2      5                                     23
-----
: : : :   :                                     :
```

Bit 0 -- Error: This bit is set by the controller when an error is detected (see below).

Bit 1 -- List full: This bit indicates that the linked buffers following the URC contain valid data. Its interpretation depends on the operation.

On a "send" operation the controller expects to find this bit a 1, indicating valid data to be transmitted.

If the controller finds this bit 0 when a "send" is initiated, the "need-new-list" bit will be set to 1 and a "send" interrupt issued.

When the "send" operation is completed the controller resets this bit to 0.

On a "receive" operation the controller expects this bit to be a 0, indicating that the buffers are ready

to receive a message.

If this bit is found to be a 1 when a "receive" operation is begun, the "need-new-list bit" will be set and a "receive" interrupt issued.

This bit is set to 1 by the controller at the completion of a "receive" operation.

Bit 2 -- Need new list: This bit is set by the controller to indicate that the "list-full" bit was not correct at the beginning of an operation .

Bits 5-23 -- End of message: These bits are set by the controller at the end of a "send" or "receive" operation.

At the end of the "send" operation these bits point to the last word of the last buffer transmitted. This is the zero pointer that terminated the transmission.

At the end of a "receive" operation these bits point to the last word filled with data from the received message.

Bits 3-4 are not used.

Second Word: The second word (URC+1) contains error codes and is described below.

F. Interrupts

Two interrupts are used by the controller, one for "send" and one for "receive."

At the normal or error termination of either a "send" or "receive" operation the respective interrupt is issued.

G. Errors

Errors are detected by the controller for both "send" and "receive" operations, and error codes are written into the words following the "send" and "receive" URCs respectively. The "IMP down" error applies to both "send" and "receive," but is reported as a "send" error only.

"Receive" errors are reported in the word immediately following the "receive" URC. The errors and bit locations in the error word are:

Bit 19 -- Message too long: The message has exceeded the maximum length of 8096 bits.

Bit 20 -- IMP does not respond: During the transmission of a message the IMP pauses for more than 100 milliseconds between bits.

Bit 21 -- List space exceeded: Space in the linked buffers has been exhausted and there are more bits in the message from the IMP.

Bit 23 -- IMP was down: Prior to this message the IMP was down, as indicated by the "IMP-down" line.

"Send" errors are reported in the word immediately following the "send" URC. The errors and bit positions are:

Bit 19 -- Message too long: The message has exceeded the maximum length of 8096 bits.

Bit 20 -- IMP does not respond: During the transmission of a message the IMP pauses for more than 100 milliseconds between bits.

Bit 22 -- IMP-ready line is down: This error is reported only when the controller is active -- that is, after a "send" or "receive" EOM has been issued and before the completion of the indicated operation.

Bit 23 -- IMP was down: Prior to this message the IMP was down as indicated by the "IMP-down" line.

V Precision Clock

A. General Information

The ARC clock system uses a high-stability Hewlett-Packard Model 105B quartz oscillator to drive two accumulators. The accumulators are:

- (1) An absolute-time accumulator with an output of year, month, day, hour, minute, and second, updated once each second

(2) A relative-time accumulator which consists of a 24-bit binary counter. This counter is advanced once each millisecond.

The short-term jitter of both the absolute and relative accumulators is 10 to 20 milliseconds. This jitter is caused by the variation in the amount of time required to access the 940 core memory.

The error caused by the oscillator drift rate is less than 1 second every 250 days.

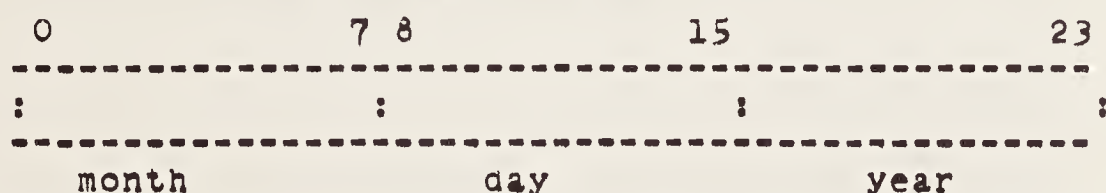
The initial setting of the absolute time is accurate to within 1 second.

The programmer has no control over the operation of this unit. Time is written in core whenever the system is operative.

B. Word Formats

The absolute time is written once each second into two words of the 940 computer.

The format of the first word is:

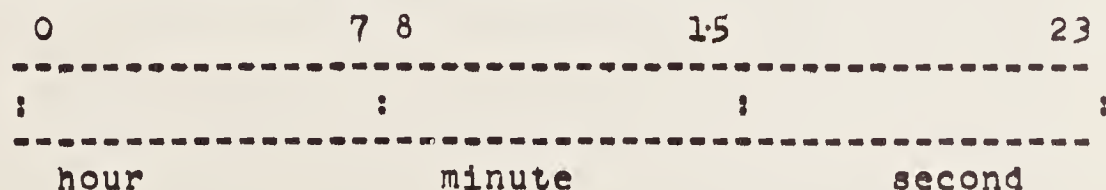


Bits 0-7 contain the month code in straight binary with a range of 1 to 12.

Bits 8-15 contain the day code in straight binary with a range of 1 to 31.

Bits 16-23 contain the year code in straight binary with a range of 9 to 99.

The format of the second word is:



Appendix C
REFERENCE MANUAL FOR PERIPHERAL EQUIPMENT

Bits 0-7 contain the hour code written in straight binary with a range of 0 to 23.

Bits 8-15 contain the minute code written in straight binary with a range of 0 to 60.

Bits 16-23 contain the second code written in straight binary with a range of 0 to 60.

The relative time is written once each millisecond into a fixed address. Bits 0-23 contain the relative time in straight binary code with a range of 00000000 to 77777777 (octal).

Appendix D
TECHNICAL DESCRIPTION OF NLS

Contents

I	Introduction.....	201
II	Utility Routines.....	203
	A. Overlay System in NLS.....	203
	1. General.....	203
	2. Implementation.....	203
	B. NLS Random-File Structure and Handling.....	204
	1. General Considerations.....	204
	2. File Structure.....	205
	3. File Handling.....	211
III	Command Specification.....	217
	A. Command Specification in NLS.....	217
	1. General.....	217
	2. Registers in the Command Specification Language.....	217
	3. Entity Character and Entity String; Command Groups.....	218
	4. Command State.....	219
	5. Command Parsing.....	220
	6. Parameter Specification.....	223
	7. Subroutine Calls and Parameter Passing.....	225
	8. Input Machinery.....	227
	9. Output (Display) Machinery.....	230
	B. Command Specification in TODAS.....	233
	1. Command Feedback.....	234
	2. Input Machinery.....	234
	3. Printing.....	236
	4. Parameter Specification.....	237
IV	Command Algorithms.....	239
	A. Editing.....	239
	1. Text Editing.....	239
	2. Structure Editing.....	248
	3. Graphics Editing.....	250
	B. View Control.....	252
	1. Jumps and Links.....	252
	2. Sequence Generator.....	253
	3. Display Parameters.....	255
	4. The User's Content Analyzer.....	256
	5. Keyword System.....	256
	6. Text Display.....	258
	C. Calculator.....	262
	D. Processors.....	264
	1. File Cleanup.....	264
	2. File Compaction.....	267
	3. Output Processor.....	267
	4. Compilers.....	267

I Introduction

This appendix gives a technical description of NLS and extends the overview given in Sec. IV-E of the main body of this report, covering the utility routines, command specification, and command algorithms used by NLS.

In addition, the special-purpose languages (SPLs) for command specification, content analysis, and string construction, which are used in large sections of NLS, are discussed in some detail.

This appendix assumes that the reader is familiar with NLS from the user's viewpoint to the level of the NLS's User's Guide.

II Utility Routines

The utility routines in NLS fall into two categories, dealing with the overlay system and with file handling.

The routines in the overlay system provide mechanisms for changing the collection of pages of code in the address space of the program; the file-handling routines provide mechanisms for referencing and changing the actual data base.

A. Overlay System in NLS

1. General

The logical structure of the overlays in NLS is a tree structure, with the most widely used code residing in the overlays near the root.

An overlay is confined to a single page, in order to make efficient use of the paging mechanisms of the 940.

2. Implementation

The overlay structure is implemented through two tables and several procedures which use them to manipulate the relabeling.

For a given page of program, there is an entry in each table. The index of the entries for the page is the same in both tables and is called the "overlay number" of the page.

One table gives the relabeling byte for the page, while the other gives the overlay number of the parent overlay and the position in the address space that the page should occupy.

The entries in the second table have a POP code in addition to the other information. To relabel in an overlay (and the overlays above it in the tree), the instruction corresponding to that overlay in the second table is executed.

If a call is to be made to a procedure in another overlay that occupies the same logical position in the address space as the calling routine, the call is split into two instructions.

These correspond to the execution of two POPs, the first of which "selects the overlay" and the second of which gives the address to branch to in that overlay.

Two cells are used in the program to keep a copy of the relabeling.

When an overlay is selected, the overlay tables are used to update these words without changing the actual relabeling.

This change is made when the second POP is executed and after the destination address has been read.

On a call such as this, the overlay number of the calling routine, as well as the calling address, is saved on a stack.

This allows the overlays to be restored to their status before the call when the called routine returns.

The routines that change the relabeling are in the overlay at the root of the tree, and are thus always available.

In general the root overlay contains utility routines for basic functions, such as changing relabeling and accessing elements of the file.

B. NLS Random-File Structure and Handling

1. General Considerations

The present format and structure of NLS files was determined by certain design considerations.

It is desirable to have virtually no limit on the size of a file. This means that it is not practical to have an entire file in core when viewing it or working on it.

A goal in the design was to make the time required for most operations on a file independent of the length of the file. That is, small operations on a large file should take roughly the same time as on a small file. In this way the user and the system are not penalized for large files.

The system had to include graphic statements, and perhaps other forms of data, as well as text.

As a result of these considerations, a random-file scheme was chosen. Each file is divided into logical blocks that may be accessed in a random order. There are several different types of blocks, and each type has its own structure.

2. File Structure

An NLS file is made up of a header and up to a fixed number (currently 66) of 1024-word file blocks.

a. The Header Block

In each file, there is a header block that contains information about that particular file.

The header block remains in memory while the file is in use.

The header includes the following information:

- (1) General information regarding the file, such as the following:
 - (a) The date of creation of the file
 - (b) The file owner's user number (identifies the user who created the file)
 - (c) The number of words in the file header block
 - (d) The initials of the user who last wrote the file out
 - (e) The date and time at the last writing
 - (f) The name-delimiter characters
 - (g) The average length of statements in characters
 - (h) The total number of statements generated in the life of the file.
- (2) Status tables for the file blocks.

The first and largest status table is the random file block status (RFBS) table.

Each entry in the RFBS table corresponds to a random file block, and indicates the status of that block. The file header is file block zero. The number in the RFBS entry has one of the following meanings:

ZERO: The block is not allocated, and does not exist.

POSITIVE: The block is allocated, and is in memory rather than on the secondary storage device. The positive number is the actual starting address for the block.

NEGATIVE: The block is not in core. If the entry equals -1, then the block is allocated, but has not been initialized. In the case of text blocks, -2 indicates that the block contains no garbage statement data blocks, and need not be garbage-collected. Otherwise the number is the negative of the used-word count.

A given file block has only one type of information, such as structure or text. There is a separate status table for each type of file block. These are called secondary status tables.

An entry in such a table has one of the following meanings:

ZERO: The block is not allocated.

NON-ZERO: The value is the block number, that is, the entry into the RFBS for that block.

There are secondary status tables for structure, text, graphics, and keyword types of file blocks. The internal structure of these different types of blocks is discussed in the following sections.

The use of separate status tables avoids references to absolute locations in the file and reduces the number of bits required to specify the location of a particular piece of information.

Pointers to various elements (structural, textual, etc.) consist of two fields: a secondary status-table index and an address giving the start of the element relative to the start of the block. The status table entry contains the number of the block, from which its absolute address can be computed.

Fewer bits are required, since the range of

secondary status-table indexes is smaller than the range of possible file-block numbers. The greatest gain from this is in the identifier for a ring element, since a file can have only eight structure blocks in the current configuration of NLS.

In spite of this, the use of the separate status tables is of questionable value.

Value of Avoiding Absolute Addresses: By avoiding absolute addresses in the file it is possible to move a block to a new location in the file simply by changing a status-table entry. Such a move can be valuable if the file has become sparse and needs to be compacted.

If absolute addresses were used, then all references to the block would have to be changed, but it can be argued that such a process need only be done on rare occasions and hence its efficiency is not crucial.

Moreover, sufficient backpointers exist so that the process of modifying references would not be difficult (although it might be lengthy).

Value of Fewer Bits in Pointers: The economy of bits in pointers is a stronger argument for the use of secondary status tables. However, the total savings per ring element (with the current size limits on files) is only six bits.

Disadvantages of Secondary Status Tables: Space in the data page is used by the tables (which are always in core) for information that would not be necessary if absolute addresses were used.

Their use places arbitrary limits on the number of file blocks of a particular type.

For example, it is possible to exhaust the structure blocks when the file actually contains room for more information. If absolute addresses were used, then blocks of a particular type could be allocated as needed, with a limit only on the total number of blocks rather than a limit on each type of block.

If further consideration confirms that the secondary status tables should be eliminated, it will not be a difficult task because of the methods used for accessing information in the files.

These methods are discussed in a later section; first the remainder of the file structure must be described.

b. File-Block Format

Each random file block has an eight-word header. This header contains the following:

(1) The checksum of the block

This is computed before the block is written, and verified when the block is read. In addition, if room in core is needed for a block, then any block in core that has not been changed may be overwritten without copying it to the file. The checksum provides an easy means of testing whether the block has been changed.

(2) The used-word count (always greater than the header size)

(3) The block type, to indicate whether the block is text or structure

(4) In structure blocks, the free-list pointer; in text blocks, the garbage-collection flag, indicating whether there are garbage SDBs (statement data blocks) in the block.

(5) The secondary status-table index number.

c. Structure Blocks

The internal structure of NLS files is a ring structure representing a tree structure. Each node in the ring corresponds to a statement, and contains pointers to the "first son" (called the sub) and the "first brother" (called the successor). The last node in a list contains a flag marking it as the tail and points to the father of its successor.

The nodes in the ring are kept in four-word ring

elements.

Each structure block contains 256 ring elements. There can be up to eight structure blocks in a file, but not all need be allocated.

Each ring element in an allocated block either is associated with a statement in the structure of the file or is on the free list for the block.

A free list consists of a chain of pointers, starting in the block header and ending with a zero pointer. (As used here a pointer is an address relative to the start of the block.) The pointers are in the first word of the four-word element, and the other three words are zero.

A free list is entirely contained within a single block in order to minimize file references.

A ring element associated with a statement contains the following information:

- (1) Flags indicating whether the statement
 - (a) has a name or not
 - (b) has been tested against the current content-analyzer pattern
 - (c) passed the pattern, if it has been tested
 - (d) is the head of its plex
 - (e) is the tail of its plex
- (2) A pointer to the text for the statement
- (3) A pointer to the picture associated with the statement if there is one
- (4) A pointer to the sub for the statement (or a pointer to the statement itself if there is no substructure)
- (5) A pointer to the successor for the statement

(6) The hash of the name of the statement if it has name.

A ring element is pointed to by a permanent statement identifier (PSID).

This is an 11-bit integer between 0 and 2047.

The three high-order bits give the structure-block number (entry into the RSVST table), and the eight low-order bits determine the location within the block.

The PSID of a statement remains unchanged as long as that statement is in the file. That is, the PSID is not changed by textual or structural editing of the file. When the statement is deleted, that same PSID may later be used to identify a different statement.

Every file has at least one ring element in its structure, namely the element for the origin statement (root of the ring, first statement in the file), which always has PSID zero.

d. Text Blocks

In addition to the header, a text-type file block is made up of an arbitrary number of statement data blocks (SDBs) and an area of free storage.

The free storage area at the end of the file block is simply a number of words available for use in creating new SDBs.

An SDB is a variable-sized block of words with a six-word header.

The header contains the following information:

- (1) The number of words in the SDB.
- (2) A flag indicating whether the SDB is unused (i.e. garbage to be collected by the garbage collector)
- (3) The PSID of the statement
- (4) The date and the time when the SDB was created

and the initials of the user who created it

(5) The number of characters in the statement

(6) The position of the first character in the statement that is not part of the name. (Set to 1 if the statement does not have a name.)

The words following the header contain the text of the statement, three characters per word. The text includes an end character (code 377B) on each end of the statement. The last word is filled to a word boundary with end characters.

The characters in a statement are explicitly numbered, the first end character being number zero.

A two-word entity consisting of a PSID and a character count is called a T-pointer, and indicates a particular character within the file.

A T-string is a string of text within a single statement.

The text-editing routines make use of T-pointers and T-strings.

e. Graphics Blocks and Keyword Block

The format of the information stored in these blocks will be described in the sections dealing with the vector package and the keyword system.

3. File Handling

a. Core Tables and File Input/Output

The random files are read into core by blocks. Two pages in NLS are logically divided into four 1024-word sections to contain the file blocks. Thus, up to four file blocks may be in core at a time. When a file block is requested, if all four are in use, one block will be written out. Core blocks may be "frozen" in, however, so that they will not be removed.

A single procedure called LODRFB controls all file input/output (other than file copying). When any routine wants a block loaded, it calls this procedure with the number of the desired block. The block is then loaded

and its location in memory returned.

The procedure makes use of several tables.

One table indicates which file block is in each core block (it is called RFIFCB for "random file index for core blocks"). A zero in this table means that no file block is there, while a positive number is the random file block number (index to RFBS).

A second table indicates which of the core blocks have been frozen. "Frozen" indicates to the file block loading procedure that the core block must not be changed. This is the case if some operation, such as editing, is being performed on data within the block.

A value in the table of -1 means that the block is not frozen; this value is incremented by 1 for each reason why the block is frozen.

The algorithm of LODRFB is approximately as follows:

First, a core block is chosen. A quick scan of the first table mentioned above is made to find an unused block. If all are in use, then a counter is used to find the next core block that is not frozen. (If all are frozen the system aborts.)

The counter provides a simple algorithm for determining which block should be removed from core.

If the chosen core block contains a file block, then one of the following things happens:

(1) If the file block is empty, it is released to the system and the corresponding status block is set to indicate that that block is unallocated.

(2) Otherwise, the block is written out on the file if the checksum has changed, and the random file status block is set to indicate that the block is on the file and not in core.

At this point the desired file block is loaded into

the core block.

If the random file block has not been initialized, the initialization is done now. Otherwise the checksum and file type are checked. An error is reported if either of these checks fails.

Finally, the random file block status is set to show that the block is now in core, and the index for core blocks (RFIFCB) is set to indicate which random file block is in that core block.

b. File Copying

The algorithm for copying an NLS file is as follows:

First, the procedure must obtain a core block to do the copying. RFIFCB is scanned to find a block that is not used. If there is no unused block, then the first block that is not frozen is taken, and the file block number in it is saved. That block is checksummed and written out on the output file (in the proper file block).

Having obtained a block, all of the allocated file blocks (except for the one already written in the event that no core blocks were free) are copied from one file to the other. This includes the file header.

Finally, if no blocks were free, the block which was removed to make room for the copy is restored from the output file.

c. Referencing Information in the File

As much as possible, information in the file is referenced indirectly through utility functions. This ensures that the file structure can be modified with minimal changes in the system as a whole.

For each field in the ring element, there are procedures which, given a PSID as argument, either read the contents of the field or store a new value into it.

Only these procedures need know the actual format of a ring element. Thus only these procedures need be changed if that format is modified.

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. II: Utility Routines

There are also procedures for reading and writing characters in an SDB. This serves both to ensure flexibility in the format of the SDB and to avoid multiple procedures for performing a very common function.

Because of the lack of instructions for character manipulation on the 940, a rather elaborate method is used to read characters from a statement.

Before any characters are read, the procedure FECHC1 is called to initialize a work area. It is called with the address of the work area and the direction in which characters are to be read from the statement.

When calling FECHC1, the first two cells of the work area must contain a T-pointer for the first character to be read. A character count of one indicates the first character of the statement. FECHC1 will initialize the rest of the work area, which contains the following:

WORD 0: PSID

WORD 1: character count

WORD 2: return address for routines reading characters

WORD 3: instruction to branch indirectly through the fourth, fifth, or sixth cells of the work area

WORDS 4, 5, and 6: address of code to pass the first, second, or third character respectively of the current word of text

WORD 7: address of the current word of text

WORDS 8, 9, and 10: the first, second, and third characters in the current word of text

WORD 11: unused

WORD 12: the address of the start of the first word of text in the SDB.

After the work area has been initialized by calling

FECHC1, any number of characters may be read from the statement by simply executing a call to the second cell of the work area. After returning the last character of the statement (or first if the direction of readout is backwards), end characters (code 377B) will be returned from all subsequent calls.

The call to the work area places the return location in the second cell and causes the instruction in the third cell to be executed. This results in a branch to a routine which returns the next character.

When all the characters from a particular word have been read, the next word of text is unpacked into the appropriate cells in the work area.

Whenever a character is read, the branch instruction in the third cell of the work area is modified so that the next call will result in a branch to the appropriate routine to read the next character.

To change position within the statement, change direction, or read from a different statement, the work area must be reinitialized by calling FECHC1 again, as described above.

Finally, statements may be read in sequence according to view parameters by means of a group of procedures collectively called the "sequence generator." This is described in detail in Sec. IV-B-2 of this appendix.

It was mentioned above that it would be possible to eliminate the secondary status tables without an undue amount of effort.

It should be evident now that this is in fact the case as a result of the use of functions to reference information in the file.

It would be possible to modify the field sizes in the ring element by simply rewriting the routines that access the affected fields.

In addition, a simple process could be written to take

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. II: Utility Routines

files in the current NLS format and convert them to a format using absolute addresses for pointers rather than status tables.

III Command Specification

A. Command Specification in NLS

1. General

The command specification section of NLS is implemented in an SPL designed to facilitate its description and implementation.

The details of this language and its use in NLS are explained in the following sections.

2. Registers in the Command Specification Language

Two types of registers are used by the command specification machinery: string registers and character registers.

Some of the registers are used internally in the implementation of the language, some are used as special-purpose registers for operations on certain types of operands, and some are general-purpose operand and storage registers.

Constructs in the input-feedback SPL allow manipulation of the string and character registers.

The principal defined operations for string registers are LOAD and DISPLAY.

The contents of a string register are normally designated in the SPL as the name of the register immediately followed by an asterisk (*).

A register may be assigned a value by a statement of the form

register-name "*" "*" expression.

Examples of expressions are:

- (1) The name of any of the string or character registers
- (2) The designation of a character, such as SP for space
- (3) The character 0, meaning to set the string to null
- (4) A string of text delimited by T-pointers.

For example, LIT*0 clears the literal input

register, while LIT*=(B1 B2) loads it with the a text string.

The contents of a register may be displayed in the name area by the command of the form

"DN(" register-name "*" ")".

Thus DN(STN*) causes the contents of the statement name register to be displayed.

The input character register is normally available to the SPL programmer as a read-only register, which always contains the last character read from the input string.

The contents of the register may be put into a string as described above, or displayed in the text area by writing DT(C*).

In addition, the input character is implicitly referenced in the case statement (described in Sec. III-A-5 of this appendix).

3. Entity Character and Entity String; Command Groups

The commands in NLS are classified in groups, and with each group is associated a particular entity (such as character, word, statement, or branch).

With this entity is associated a character called the "entity character" and a string called the "entity string."

The entity character is programmatically assigned values in the SPL by the construct

"E*=" character "," string.

This causes the entity character to be set to the value of the character, and assigns the value of the string to the entity string.

Thus "E*=B,BRANCH" sets the entity character to "B" and the entity string to "BRANCH."

The entity string and entity character are used to provide a default option in command specification.

When the command operation (such as DELETE) has been specified, the entity string for the group of the operation is offered as the type of entity for the command. The user may accept this by typing a "command accept" character (CA) or specify some other entity by typing the appropriate character.

The actual SPL constructs used to express this use of the entity string and entity character are presented in a later example.

4. Command State

Except when a command is being specified or executed, the user is in some command state.

If the user begins parameter specification without first specifying a new command, the command executed will be that designated by the current command state.

The command state is defined internally by a special register called the "state register."

The state register always contains the location of the most recently defined command state.

This location is in the same format as a return location placed on the stack in a subroutine call.

The state register additionally contains the command group of the command state.

The SPL syntax for defining a command state is

"S*=" label "," command-group,

which results in a call to the state defining routine to be produced by the compiler. The label is defined as being equal to the address of this instruction.

From the command state, control passes directly to a parameter specification point in the program, which acts as an idle or "wait for next input" point.

Control returns to the highest level of the command parsing code if the character read is not a legitimate parameter specification character.

This is one of the most significant features in making the command language efficient and easy to use.

The contents of the state register may be used as an operand in designational expressions.

Thus, one may programmatically return to the previous command state by the SPL statement "GOTO [S]".

There are several occasions where this construct is used

At any time during the command specification, a user may return to his previous command state by typing a "command delete" character (CD).

From the above description of command state, it may be seen that the action of a command delete is to reset any parameters entered during the course of the aborted command and branch to the location contained in the state register.

If a specification error occurs during the execution of a command, the command is aborted and NLS is automatically returned to the previous command state.

5. Command Parsing

The NLS input commands are parsed through the use of nested case statements.

The depth in the nest of case statements corresponds to the position of the next character to be read in the command input string.

Thus if a command were specified by three characters, the first character would be read by a first-level case statement, the second by a second-level case statement, and the third by a third-level case statement.

Two features of the case statement construct in the input-feedback SPL make it especially suited for parsing the command input strings.

The selection criterion for the execution of an element of the case statement is equality of two specified characters, one of which appears at the front of the element, the other of which is implicit,

The implicit character is normally the last character read from the input string. In addition, it is possible to repeat a case (using a "REPEAT" construct) with some character other than the input character.

In particular, the entity character may be used. This permits the implementation of the command default option mentioned above.

At the head of the case statement, the entity string is used to offer a default value of the command type. If the user types a command accept, there is an element in the case statement which is executed and results in repeating the case statement using the entity character in place of the input character.

The net effect is the same as if the user had typed the entity character rather than a command accept.

If none of the tests succeed, then an "ENDCASE" statement is executed.

Whenever a case statement is executed, an entry is made on a stack indicating the location of that case statement.

A construct in the repeat statement allows the execution of a previous case statement with a particular character.

The word REPEAT is followed by an integer indicating which of the stacked cases is to be repeated.

Thus REPEAT 2 causes the second previous case statement to be repeated.

The integer is in turn followed by a character specification in parentheses.

This may be any of the following:

- (1) An actual character to be used, such as SP
- (2) The entity character (E*)

(3) The next input character, indicated by a period.

A brief example of code for parsing an NLS-like command language is presented here.

It incorporates most of the SPL constructs mentioned in this section, as well as some not mentioned.

The command language described here allows two groups of commands, used for text editing and structure editing respectively.

Four commands are specified:

Text editing: (initial entity = character)

Insert Character

Insert Word

Structure editing: (initial entity = statement)

Append Statement

Append Branch

(start) . case

(i) [textedit] dsp(< insert ↑ es*) . case

(c) s**ic,textedit dsp(+ < insert character)
e**c,character +parmspec,prmspc -comex,exectr

(w) s**iw,textedit dsp(+ < insert word) e**w,word
+parmspec,prmspc -comex,exectr

(ca) repeat 0(e*)

(cd) goto [s]

endcase goto start

(a) [stredit] dsp(< append ↑ es*) . case

(s) s**ic,stredit dsp(+ < append statement)
e**s,statement +parmspec,prmspc -comex,exectr

```
(w) s*=iw,stredit dsp( + < append word) e*=w,word  
+parmspec,prmspc -comex,exctr
```

```
(ca) repeat 0(e*)
```

```
(cd) goto [s]
```

```
endcase goto start
```

```
endcase repeat 0(.)
```

6. Parameter Specification

Parameter specification is that portion of NLS which is involved with the selection of operands for commands.

Operands may be specified by selecting locations and entities in a file, by entry of strings from the keyboard, or by the naming of pointers with the keyset and mouse.

Specifications of entities in the file are represented by one or more entries on a stack, called the specification stack. (This is independent of the subroutine argument and return stack.)

There is one entry on the specification stack for each selection made in parameter specification.

A normal entry on the specification stack (spec stack for short) is called T-pointer (which consists of a PSID and a character count).

An SPL construct facilitates the placing of arguments onto the spec stack. The syntax is

```
"SPEC(" argument ")",
```

where an argument can be any of the following:

BUG: Process the most recent command accept as a bug selection and place the corresponding T-pointer on the spec stack

POS: Load the last bug selection onto the spec stack.

String register: The action of this command depends on the register specified, and the contents of the

register.

If the register is the number register, then the number string in the register is converted to an integer and pushed onto the spec stack as the second word

If the specified register is the statement number register, it converts the string in the register (assumed to be a statement number) into a PSID, and pushes it onto the spec stack

In the case of any other register, if the first character in the string is a digit, then the content of the register is assumed to be a statement number, otherwise, a statement name. In either case the corresponding PSID is pushed onto the stack.

Number: The integer indicated is pushed onto the spec stack

Identifier: The value of the identifier is pushed onto the spec stack

(no argument): This causes the spec stack to be cleared of all entries.

A textual entity may be specified (effectively) only through bug selection(s) or with a pointer.

A structural entity may be specified by bug selection(s), a pointer, or keyboard entry of statement name(s) or number(s).

In the case where the bug selection or pointer serves as a text selection which indicates a string identifying the statement to be specified (e.g., names, links), the selected string is moved into a string register and treated as though it were entered from the keyboard.

The algorithms for converting bug selections into T-pointers are discussed in Sec. IV-B-6-c of this appendix.

A pointer is simply a T-pointer which has been given a name and stored in a table.

It is specified by depressing the right button on the

mouse, and entering the name of the pointer with the keyset.

When a pointer has been specified, the associated T-pointer is simply loaded into the internal register containing the (processed) mouse location, making it appear as though a bug selection had been made.

A statement may be selected from the keyboard by typing either the statement name or the statement number.

A statement number is converted into a PSID for a T-pointer by simply running through the ring at each level (beginning with level 1) until the specified statement is reached, or found to be non-existent.

A statement name is converted into a T-pointer by running through the ring, looking for a statement which has a name, and whose hash is the same as the hash of the name being searched for.

In the case where an operand is a textual entity which is entered from the keyboard, there need not be an entry on the specification stack for it.

Rather, it will go directly into a specified register, and be used in that form for the command.

It should be noted that the selections of textual entities in the file are processed during execution of the command so that (when appropriate) the textual entity is put into a register in the same form it would be in if it had been entered from the keyboard.

7. Subroutine Calls and Parameter Passing

The subroutine call mechanism in the SPL is very similar to that used by ALGOL. It uses a stack for containing return information, parameters, and local variables.

Because of the overlay structure of NLS, it is necessary to indicate in a subroutine call not only the address of the routine being called, but additionally the name of the overlay in which that routine resides.

The name of the overlay containing the calling routine is stacked with the return location, so that the appropriate overlay may be relabeled in upon return.

There are two types of subroutine calls, which differ in the return locations placed on the stack.

The return location stacked by a normal subroutine call is the address of the location following the calling instruction.

The other subroutine call stacks the return location of code which will return NLS to the previous command state.

The format and operation of the stack (and subroutine call mechanism) are roughly as follows:

The stack is addressed by two pointers, one to the current base and one to the stack top.

A subroutine call instruction is always preceded by a "mark stack" instruction.

The "mark stack" instruction pushes the contents of the base-of-stack pointer onto the top of the stack, followed by a zero (which will be used by the actual subroutine call for the return location).

The top-of-stack pointer is incremented accordingly, and the base-of-stack pointer is set to point to the new top of the stack (which will eventually contain the return location).

Formal parameters are now loaded onto the top of the stack.

If an overlay has been specified in the subroutine call syntax, a cell is set to reflect the overlay containing the procedure being called.

Note that the actual program relabeling is not changed at this time.

The subroutine call is now executed.

The return location is computed.

This is a combination of the calling address and the name of the overlay containing the subroutine call instruction.

This is true except in the case of the special subroutine call which returns to the previous command state.

In the special subroutine call, the contents of the state variable (which in fact is the return location for the previous state, as computed above) are used as a return location.

The return location is stored in the cell pointed to by the base-of-stack pointer.

Finally, the overlay containing the called procedure is relabeled in if necessary, and a branch is made to the address indicated in the subroutine call.

The syntax of a subroutine call in the SPL is

("+" / "-") procedure-name ("," overlay-name / EMPTY),

where " / EMPTY" means the construct before the slash is optional.

In addition, parameters may be specified by listing them in square brackets after the call. Individual parameters in the parameter list are separated by commas.

The "+" indicates a normal subroutine call, and a "-" indicates a special subroutine call which returns to the previous command state.

If no overlay name is specified, an overlay which is either the overlay containing the calling procedure or an overlay above it in the overlay tree is assumed, and thus no change is made in the relabeling.

An example of a subroutine call is

+subpat +wdr2,txtedt(bl,pl-4) -qdv,txtedt.

8. Input Machinery

a. Work Station Input from Keyboard, Keyset, and Mouse

Characters are read from the work station by a system routine in the following manner:

Whenever a button on the keyboard, keyset, or mouse changes state, the TSS I/O software considers it a character entry, and places the following information into its input buffer.

- (1) The device which caused input
- (2) A code which is the input itself:
 - (a) A character in the case of the keyboard
 - (b) A code in the case of the keyset
 - (c) A down/up and button indication in the case of the mouse
- (3) The mouse coordinates at the time the character was read
- (4) The time (16 millisecond resolution) when the character was read.

A system call is then used by NLS for reading the characters from the system input buffer, which returns a character (and related information as described above) if there is one, and reports the status of the system input buffer (empty, another character waiting in input buffer, no character read).

b. Input Fork

Because of the necessity to read characters from the system input buffer so that it does not overflow -- and more important, to provide a facility to interrupt NLS while it is executing a long process -- a fork is activated to run asynchronously in parallel with NLS.

This fork may be conceptualized as an independent program (called the input fork) which reads characters from the work station and places them in a programmatic input buffer to be read later by NLS.

NLS always reads characters from the programmatic input buffer before reading them from the system, and when it is reading a character from the system, it checks to ascertain that the input fork is not reading the same character.

The input fork additionally has the capability to interrupt NLS from the process it is currently involved in, and it does so when it reads an interrupt character (RUBOUT) from the keyboard.

Since NLS always reads characters passed to it from the input fork before reading those waiting in the system, and there is no restriction on where the input fork gets the characters it will pass to NLS, the input fork may be used to simulate an NLS user.

A simple facility is currently provided along this line, whereby the input fork can read characters from a file, and (with a minimum of translation and interpretation) pass them on to NLS.

This feature is used mostly for merging and converting sequential files into NLS files.

c. Character Translation

The keyset and mouse input requires translation from its raw input form to a character which is meaningful to NLS.

The keyset input is in the form of a number (0-31) which reflects the keys depressed (and released) on the keyset.

This is combined with the current state of the left and middle mouse buttons (which provide a case shift) to produce the translated character.

The translation algorithm is roughly as follows:

If both mouse buttons are down (case 3) then this is a view specification character, so treat specially.

Otherwise, use the keyset character as an index into a table of character values.

This table of character values has three entries for each possible keyset value, one for each of the remaining cases.

The case is then used to determine the correct table entry as the translated character.

Additional translation is done when characters are entered from the mouse without concurrent entry from the keyboard or keyset.

This translation simply looks for combinations of up/down strokes of mouse buttons without intervening characters, and translates them to specific characters.

This is used for the command accept, command delete, backspace character, and backspace word characters.

9. Output (Display) Machinery

a. General

NLS communicates with the user via a display screen divided into six areas.

Each area is maintained separately of the others, and contains a specific type of information.

The organization of the registers on the display screen, and the format of the registers themselves, are parameterized.

There are many parameters which relate specifically to certain registers, and some parameters which relate to all registers. Among the parameters relevant to all of the registers are:

location on screen

character size and type used in register

display of register on/off

Insofar as possible, these parameters are the display control words used by the hardware. This minimizes the software required for controlling the screen format.

b. View Areas

(1) Echo Register

The echo register is maintained by the system and reflects the raw character input to NLS.

NLS is concerned with this register mainly at initialization, when it must be set up by a series of system calls.

(2) VIEWSPEC Area

The view specification (VIEWSPEC) area reflects those text area view parameters which are not obvious from looking at the text area.

The VIEWSPEC area is changed by the same routine which changes the view parameters themselves.

(3) Command Feedback Line

The command feedback line is the major feedback mechanism of the command specification machine.

There are two components in the command feedback line: words which reflect in English the command being specified, and an arrow which indicates the user's state in specifying the command (the arrow most commonly indicates whether the user may specify a new command or parameters, or whether he is currently specifying an entity).

There are three possible positions to which a word may be moved in the command feedback line:

First position: This causes the command feedback line to be cleared, and the designated word to be displayed as the first word in the line.

Next position: This appends the designated word to the end of the command feedback line.

Last position: This replaces the last word in the command feedback line with the designated word.

The arrow may be pointed to the beginning of the word in a specified position in the command feedback line, or it may be turned off.

The SPL construct provided for the manipulation of the command feedback line is

"DSP(" display-parts ")",

where the syntax of a display-part is

word / "ES*" / "<" word / "... " word / "+" / "↑".

The DSP command rearranges the command feedback line so that it is formatted in accordance with the display-parts.

The meanings of the display parts are as follows:

Word: A string equal to the text of the the word i placed in the indicated position in the command feedback line

"ES*": The contents of the entity string are displayed in the indicated position in the command feedback line

"<" word: The word is placed at the left of the command feedback line

"..." word: Replace the last string in the current command feedback line with the word

"+" : Position the up-arrow to the front of the command feedback line.

"↑" : position the up-arrow at the start of the following string in the command feedback line.

There are three additional intrinsic functions which are used in relation to the command feedback line. These are

AF Turn off display of arrow

AN Turn on the display of the arrow

QM Display question mark beside the arrow.

(4) Name Register

The name register is used for displaying statement names and arbitrary strings relating to parameter specification.

An SPL function is provided which moves the contents of an arbitrary string register to the name register.

The syntax is "DN(" register ")".

(5) Date/Time Register

The date/time register always reflects the date and time.

It is updated every 10 seconds by a fork (similar to the input fork in its relation with NLS) whose sole job is to read the date and time from the system, place it in a core location, and dismiss itself for 10 seconds.

(6) Text Area

The text area serves as the user's window into his file.

What is displayed in the text area is a view of the user's file, subject to certain formats and reorganization, which is described by a set of parameters (called view specifications or VIEWSPECS).

The creation of new views is programmatically caused by the display SPL construct "DISPLAY(" optional-parameter ")".

If there is a parameter, it is used to determine the PSID of the starting statement for the view creation.

The process of creating a view of the file in the text area is discussed in Sec. IV-B-6 of this appendix.

c. Literal Feedback

When a literal string is entered as a part of parameter specification, it is placed in the text area (beginning at the top) according to the format of the text area.

The part of the file view which was previously in the space used by the literal feedback is temporarily replaced by the feedback.

B. Command Specification in TODAS

The TODAS command specification system is much simpler than

that of NLS, insofar as it does not use the state machine and no command state is defined other than the null command RESET.

1. Command Feedback

The command language input string is parsed by case statements in a manner similar to NLS.

The command feedback may best be described as complex character echoing, where each command specification character is reflected by the typing of appropriate words and the state of the command specification is indicated by the position of the carriage.

As in NLS, the user has the ability to control parameters relating to the command feedback, including the number of characters of each word echoed.

2. Input Machinery

Much of the NLS input machinery is used by TODAS.

There are, however, some differences:

Because of the allowance which the system makes for an interrupt character (RUBOUT), and the fact that the system teletype buffers are larger than the system work station buffers, an input fork is not required.

One may still be used, however, in special cases such as sequential file input.

All characters read by TODAS undergo a translation on input.

This facilitates the effective interfacing of TODAS to a number of input devices (six different types of typewriter terminals are currently provided for).

The character translation is accomplished by a table look-up technique (the table is indexed by the raw character value).

The result of the look-up may be a normal text character, or it may be a special character (which is indicated by the high-order bit).

In the event that it is a special character

(command accept, command delete, shift character, centerdot, etc.), an appropriate action is taken if necessary. The character may be echoed (as some previously designated character), and it may be specially flagged as a control character.

There is, in addition to straight character translation, a facility to define shift characters which allow devices with restricted character sets (e.g. upper case only) to work with full character sets.

Four shift modes are currently defined in TODAS:

Null: No shifting takes place

Mode 0: Upper-case alphabetic characters are translated to lower case

Mode 1: Lower-case alphabetic characters are translated to upper case

Mode 2: Lower- and upper-case alphabetic characters are translated to control case

TODAS is in one of these modes (as a base mode) at all times.

The mode may be changed (either temporarily or permanently) by typing a character which has been defined as a shift character for the new mode.

There are currently three types of mode-shifting characters:

Character shift: This causes the following character to be translated according to the mode for which the shift character has been defined, if it is a character which would normally have been translated in either the base mode or the shift mode. If the character would not have been translated, then the shift character is treated as a normal character.

Word shift: This causes the following word to be translated subject to the same rule as given above for character shift -- i.e., if

the next character is translatable, the word is translated; otherwise the shift character is treated as a normal character.

Permanent shift: This causes the base mode to be changed, and all subsequent characters are translated according to the new mode.

The shifting is accomplished in the following manner:

If, a permanent shift character is read at any time, the shift mode is changed and another character is read normally.

If a word-shift or character-shift character is read, the next character is read from the input string.

If the next character is a shiftable character, then the shifting is performed, and the shifted character is the result.

If the shift character is for a word shift, then a global parameter indicating the current shift state is set accordingly, and will not be reset until a space is read.

If the next character is not a shift character, it is returned to the front of the input string and the shift character is returned as a normal character.

3. Printing

Printing of a structure in TODAS is analogous to creating a new view for the text area in NLS, insofar as the same view specifications are used for interpreting and formatting the file.

Three differences are apparent:

The text area is of unlimited length, so that a whole file may be seen in one view. Pagination is performed when a long view is created.

Text undergoes an output translation and shifting

which is a counterpart of the translation and shifting done on input.

The user has a degree of interactive control over the view being created, specifically:

The creation of a view of any particular statement may be aborted at any time.

The creation of the entire view may be aborted at any time.

Implementationally, formatting routines different from those used by NLS are employed.

The output is formatted one line at a time, and the printing of an entire statement must physically finish before the first line of the next statement will be printed.

This restriction is necessary because TODAS must know which statement is currently being typed in order to respond properly to the user's request to abort the view of the statement.

The same sequence generator is used, but the structure being printed is searched one branch at a time (except in the case of trails and keyword).

4. Parameter Specification

Parameter specification differs from NLS in three important ways:

All specification must be done via the keyboard.

A "current statement" is defined as an operand at all times.

The execution of any command without a specified operand assumes this statement as an operand.

The current statement is represented internally as a cell containing the PSID of the last statement addressed in the successful execution of a command. It is updated each time a command is successfully executed.

The one exception to this is that during printing, it is set by the print routines to the PSID of the last statement printed.

Operands (statements) may be addressed relative to each other in the tree structure of the file.

For example, one may specify a statement which is the "successor of the down of the tail" of the current statement -- i.e., the successor of the first substatement of the last statement in the same plex at the same level as the current statement.

The relative addresses of operands are interpreted as they are entered by accessing the ring (as necessary). Any error is reported immediately, and nullifies the entire address (except in the case of links).

Links are parsed whenever they are referenced in an address field, and executed immediately after selection. That is to say, when a link is encountered in an address field, the current statement is changed immediately to reflect the value indicated by the link.

IV Command Algorithms

A. Editing

Editing in NLS includes textual, structural, and graphical modifications to the file.

The textual and structural editing actions include insert, move, replace, delete, and copy. These actions may be performed on textual entities such as characters, words, and visible strings, as well as structural entities such as statements, branches, groups, and plexes.

The graphical editing actions include insert and delete for vector labels, and insert, delete, move, transpose, and vertical and horizontal projection for vectors.

1. Text Editing

a. General Considerations

The process of textual editing will be discussed first. This process basically consists of delimiting the appropriate substrings, by means of the content-analysis SPL, followed by construction of one or more new statements with the desired modifications. This latter step is specified by a procedure written in another SPL, the string-construction SPL.

These content-analysis and string-construction procedures are written in such a way that in spite of the large number of combinations of editing actions and textual entities, there is a single content-analysis procedure to delimit each entity and a single string-construction routine to perform each action.

This is done by standardizing the way in which a substring is delimited by the content-analysis procedures.

Four pointers are passed to the procedure as arguments, along with one or two selections made by the user.

When the procedure returns, the appropriate substring is delimited by the pointers in the following manner.

The first and second pointers mark the first and last characters of the substring, respectively. The third and fourth pointers mark the characters to the left and right of the substring, respectively.

Thus if P1, P2, P3, and P4 are the arguments, the characters from the front of the statement up to P3 precede the desired substring, the characters from P1 to P2 are the substring, and those from P4 to the end of the statement follow the substring.

A detailed description of the word-delimiter routine is useful to clarify this process.

There are five arguments; the first is the position of the user's selection, the remaining are pointers to be used to delimit the actual text of the word in the manner described above. The body of the procedure is simply

```
a1 > CH $LD ↑a3 ↑a5 ←a3 a1 < CH $LD ↑a2 ↑a4 ←a2
```

which has the meaning "starting from the selection (a1) scan to the right (>) past a character (CH) and any number of letters or digits (\$LD). Set a3 and a5 to the resulting position (↑a3 ↑a5) then move a3 back (←a3) so that it points to the last character of the word. Now reset the search pointer to the selection (a1) and scan to the left (<) to set a2 and a4 (↑a2 ↑a4 ←a2)."

Once the substrings have been delimited in the above manner, new statements are constructed under the control of procedures written in the string-construction SPL.

The syntax of a statement in the string-construction SPL is as follows:

```
scstat = "IF" posrelation "THEN" scstat "ELSE" scstat  
/  
"BEGIN" scstat $(";") scstat) "END" /  
  
"ST" pos "+" pairlist;
```

The position and position-relation constructs are the same as in the content-analysis SPL.

A pairlist is a list of pairs, in this case separated by commas.

A "pair" specifies a string of text, usually by giving two positions which delimit the string.

In addition the "pair" can be a constant string or the contents of some variable string such as the literal input register.

The meaning of "ST pos ← pairlist" is "The statement pointed to by pos is constructed from the strings specified by the items in the pairlist."

Thus, assuming that the pointers have been set as described above, "ST B1 ← SF(B1) P3, P4 SE(B1)" would cause the text from P1 to P2 to be deleted from the statement selected by B1.

The "move" procedure offers a more complex example. The procedure has ten arguments; a1 and a2 are the user's selections, a3 through a6 are the pointers associated with a1, and a7 through a10 are the pointers for a2. The body of the move routine is

```
IF SF(a1) = SF(a2) THEN BEGIN
  IF a1 < a2 THEN
    ST a1 ← SF(a1) a4, a7 a8, a6 a9, a10 SE(a1)
  ELSE
    ST a1 ← SF(a1) a9, a10 a4, a7 a8, a6 SE(a1) END
ELSE BEGIN
  ST a1 ← SF(a1) a4, a7 a8, a6 SE(a1);
  ST a2 ← SF(a2) a9, a10 SE(a2) END
```

The pair a7 a8 delimits the text to be moved. The positions a9 and a10 will become adjacent when the text from a7 to a8 is moved. The destination of the text between a7 and a8 is after a4 and before a6. The reader should convince himself that the above procedure does this in all cases.

b. Implementation

The code compiled for string-construction SPL routines consists mainly of calls to MOL procedures.

At the start of the code for a pairlist there is a call to a procedure called BSC (begin string construction) and at the end of the pair list there is a call to ESC (end string construction). For the actual items in the pairlist, procedures are called which append the appropriate strings onto the statement being constructed.

The BSC procedure must create a new statement data block

(SDB) to hold the text of the statement being constructed. Since the final size of the statement is not known at the time BSC is called, the average size of SDBs in the file is used as an estimate of the number of words required for the new SDB.

The search for the required amount of room begins in the file block containing the old SDB, if there was one.

If there is not adequate room there, then the procedure looks for room in the file blocks, starting with the lowest index number.

This ensures that if there is room in a block already allocated, then that room will be used rather than causing a new block to be allocated.

The procedure ISROOM is called to determine whether there is adequate room in a given file block.

If the block is unallocated, then ISROOM returns TRUE.

If the block is allocated and contains adequate free storage, then such information is held in the status table, RFBS. This avoids the possibility of reading a file block only to find that it does not contain adequate room.

If the block does not contain adequate free storage, but does contain garbage SDBs (also known from RFBS), then ISROOM calls the garbage collector to process the block.

Garbage collection involves moving nongarbage SDBs to fill in the gaps occupied by garbage SDBs and updating pointers in the ring elements corresponding to the moved SDBs.

If this produces enough room, then ISROOM returns TRUE; otherwise it returns FALSE.

After sufficient room has been found by the above process, the BSC procedure builds a header for the new SDB and then sets up a work area for the subsequent string transfers that will take place during the construction of the statement. This work area

contains information such as the address of the SDB. This completes the tasks of BSC, and it returns.

The actual construction of the new statement consists of appending characters onto the new SDB.

For those parts of the statement that remain the same, the text is read out of the old SDB into the new. New parts of the statement are simply characters from other sources, such as literal input or other SDBs.

The observant reader will realize that it is possible to run out of room while appending characters.

If this happens, the block is garbage-collected. If this results in room for at least 60 more characters, then the SDB under construction is simply moved in with the same file block to make more room.

If garbage collection of the file block cannot produce that much more room, a location in a different file block is found that does provide the required space. The partially constructed SDB is then moved to this new location.

When all the strings have been appended to the SDB, the procedure ESO is called to finish the job.

It first gets rid of the old SDB for the statement, then does the bookkeeping to establish the new SDB as the SDB for the statement. This involves updating the SDB header, the running average length of SDB's, the pointer in the statement's ring element, and the name hash for the statement in the ring element.

In addition the "content analyzer pattern tested" flag for the statement is turned off (see Sec. II-B-2-c of this appendix).

This completes the construction of a new statement and our discussion of text editing in NLS.

c. Content-Analysis SPL

In NLS it is often necessary to analyze the textual content of a statement in order to delimit certain substrings.

For example, the user may select a word of text for editing by pointing to any character within the word. The actual substring making up the word is determined by NLS.

A special language, the content-analysis SPL, is used for writing such string delimiting procedures.

Basically, the language provides constructs for controlling the position of a search pointer in a text string and saving various positions in order to delimit the desired substrings. (In the discussion of the content analysis SPL, position refers to a statement identifier and character number -- in other words, a T-pointer as defined elsewhere.)

The initial position of the search pointer is often determined by a selection made by the user. The positions of such selections are stored in buffers B1, B2, etc.

Pointers P1, P2, ... may be used to store positions. The current position of the search pointer can be stored in Pn by writing ↑Pn.

Arguments may be passed to a content analysis procedure. Such arguments are either bug selections (i.e. Bn) or pointers (i.e. Pn). Since the procedure must be able to set the pointers to appropriate values, these parameters are called by (simple) name rather than by value. The formal parameters are A1, A2, etc.

The three forms, Bn, Pn, and An, are the basic ways of referencing a position. In addition, there are two functions taking a position as argument and yielding a position as result. These are SF and SE, which give the position of the statement front and statement end, respectively, of their argument.

The position of the search pointer can be set by simply writing any of the above forms to determine a position. For example, "SF(B1)" puts the search pointer at the first character in the statement first selected by the user.

The search pointer is also moved by tests for basic text elements. The basic text elements are strings, single characters, and character class variables.

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. IV: Command Algorithms

A string is a sequence of characters delimited by quote marks (").

If the string matches the sequence of characters starting at the current location of the search pointer, then the search pointer is moved to the next position beyond the string and a general flag is set TRUE.

If, on the other hand, there is only a partial match, or no match, then the search pointer is not moved and the general flag is set FALSE.

The test for a single character is logically equivalent to testing for a string of length one, but is implemented in a more efficient manner. The single character is specified by preceding it with an apostrophe.

The implementation of these tests makes use of the programmed operator (POP) facility of the 940.

For the single character test, the computer produces a single instruction in which the address field contains the code for the character and the rest of the instruction specifies the POP to perform the test.

Similarly, the string test results in an instruction specifying the number of characters in the string and the appropriate POP, followed by words containing the actual string.

The basic text elements of the third type -- the character class variables -- are also implemented using a programmed operator. The character class variables allow tests for any character in a particular class. The classes, with their associated variable names, are as follows:

LD	any letter or digit
L	any letter
D	any digit
NP	any nonprinting character

PT any printing character
SP space
TAB tab
CR carriage return
CH any character

These tests are implemented in a manner very similar to the single character test, except the address field of the instruction contains a class code rather than a character code.

The successful completion of one of the above tests causes the search pointer to be moved. The direction in which it is moved, towards the end of the statement or the front, may also be controlled.

A ">" means scan (move pointer) to the right, or towards the end, while "<" means scan left.

As mentioned above, the current position of the search pointer can be saved by writing "↑" followed by either Pn or An.

In addition the value stored in a buffer can be modified to point to the preceding character, according to the current scan direction, by writing "←" followed by Pn or An.

The reason for this operation is that when an entity has been successfully found the pointer is left pointing to the character beyond the entity. Thus to save the position of the last character in the entity it is necessary to write ↑Pn←Pn.

The remainder of the language simply provides for building more complex expressions from the basic text elements presented above.

One of the primary means of doing this is the arbitrary number operation. The general form of this is m\$N followed by a text expression and has the meaning "from m to n occurrences of the given expression."

Both the upper and lower bounds are optional, with default values of 1000 and 0 respectively.

This is implemented in the following manner.

The upper and lower bounds and a count, initially zero, are pushed on the stack. Then the test for the expression is repeated until it fails, with the count being incremented at the completion of each successful test.

When the test for the expression does fail, the current value of the count is checked against the bounds and the general flag set accordingly.

The other operators, in order of decreasing precedence, are as follows:

- (minus sign): indicates negation.

After the test for the text expression following the minus sign, the value of the general flag is complemented.

(space): indicates concatenation.

After the test for each element in a sequence of concatenated tests, the general flag is tested. If it is false, then the preceding element was not found and control branches to the location following the current sequence of concatenations. If the flag is true, then the next test in the sequence is performed.

/ (slash): indicates alternatives.

If the expression on the left of the slash is found, then control branches beyond the sequence of alternatives. Otherwise, the search pointer is reset to its position prior to the test for the previous alternative and the next alternative in the sequence is tested.

NOT: indicates negation.

Equivalent to minus sign except for lower precedence.

AND: indicates logical conjunction.

If the expression on the left of the AND is not found, then control branches beyond the expression on the right of the AND. Otherwise, the search pointer is reset to its position prior to test for the left expression and then the right expression is tested.

OR: indicates logical disjunction.

Like AND except branch if flag true instead of false.

Any expression built using the above operations may be enclosed in parentheses and used as a basic element in a concatenation.

Similarly, any such expression may be enclosed in square brackets and used as a basic element. The effect of the square brackets is to "unanchor" the scan. In other words, as long as the test fails, it is repeated starting one character farther along in the statement until either the statement is exhausted or the test succeeds.

Thus `["abc"]` is satisfied if the remainder of the statement contains the string "abc".

Finally, a conditional statement is included in the language to allow a pattern to be selected for testing on the basis of a comparison of positions.

If two positions are in different statements, then all relations between them are false except "not equal." Otherwise, the relationship depends on the character number of the position. For example, if B1 and B2 are in the same statement, B1 pointing to character number 3 and B2 to character number 20, then B1 is less than B2.

This completes the description of the content-analysis SPL.

2. Structure Editing

Like text editing, structure editing consists of a phase in which the entity to be edited is delimited, followed by the

actual editing action.

Since the structural entities "branch" and "plex" are simply special cases of the group entity, the editing routines all deal with either a single statement or a group.

The delimiting for the move and delete commands is the same.

In all cases a group, specified by two PSID's, is the final entity on which the editing action is performed.

For a branch the two PSID's for the group are set to the PSID of the selected statement.

For a plex the PSID's are set to the head and tail of the plex of the selected statement.

For a statement, a test is made to ensure that the statement has no substructure, after which it is treated like a branch. (If the statement does have substructure the command is aborted.)

Finally, if the specified entity is a group, then the two selected statements are checked to verify that they do in fact specify a valid group.

Once the group has been delimited, the move commands perform the following sequence of operations.

First, the destination is checked to make sure it is not within the specified group. The command is aborted if it is.

The group is then removed from the ring structure by the appropriate changes in pointers and flags in the ring element of the predecessor (and possibly the successor) of the group. The group is then reinserted into the ring in its new location through another set of changes in pointers and flags. Notice that no text is moved and no statement identifiers are changed. The only changes are in the successor and substatement fields and the head and tail flags of four or five ring elements.

The execution of delete commands naturally results in greater changes. The group is first removed as in the move operation. Then the statements making up to the group are deleted according to the following algorithm expressed in MOL.

```

d1←grp1; %start with the first statement in the group%
LOOP BEGIN
  WHILE (d2 ← getsub(d1)) NOT= d1 DO BEGIN
    %d1 has substructure%
    stosub(d1,d1); %change sub-pointer
    so that d1 no longer appears to have
    substructure%
    d1 ← d2 %more to sub% END;
  %when exit the WHILE statement,
  d2 equals d1 and has no substructure %
  d1 ← getsuc(d1); %move d1 to the successor,
  which will be back to the "father" statement
  when all of its descendents have been deleted%
  relst(d2); % release SDB for d2%
  frersv(d2); % free ring element for d2%
  IF d2 = grp2 DO-SINGLE RETURN END;
  %finished when have deleted top statement of last
  branch in group%

```

Note that since the successor of the last statement in a plex is the father of the plex, no stack is needed in the above algorithm. Also note the manner in which the sub-pointers are modified to guide the traversal of the group.

As might be expected, copying a group is more complicated than deleting one since the structure cannot be modified during the process.

In very simplified form, the copy group algorithm is as follows:

Starting at the first statement in the group, if the statement has substructure, copy that first; then copy the statement and move to its successor until the last statement in the group has been copied.

When the group has been copied, it is inserted in the appropriate position in the ring in the same manner as a group being moved is reinserted into the ring.

3. Graphics Editing

Blocks containing picture information are virtually identical to those containing text information. The main difference is the replacement of statement data blocks by vector data blocks (VDB's).

A vector data block is made up of a header and an arbitrary number of lines and labels making up a picture.

The header contains much the same information as is held in the header of an SDB. Instead of character counts, however, the VDB header contains a count of the number of lines in the picture.

Following the header is a sequence of two-word buffers, each representing a line in the picture.

The first word gives the position of one end of the line relative to the lower left-hand corner of the text of the statement.

The second word gives the position of the second end of the line relative to the first endpoint.

Following the buffers for the lines, each label in the picture is stored as a position (in the same format as the first word of a line buffer) and a text string.

The current vector package was developed on a trial basis with a relatively small programming investment. As a result of this, the only graphic entities available are lines (vectors) and labels. A more sophisticated graphics system has been designed but not yet implemented.

Selection of these entities is handled in the following manner.

Line selection is done by finding the line that minimizes the difference between the sum of the squares of the distances from the endpoints of the line to the bug selection and the square of the length of the line.

This is a practical algorithm since the number of lines involved is small (under 100).

Label selection is done by finding the label that minimizes the square of the distance between the bug selection and the second character of the label.

The "move vector" command will be explained as an example of vector editing.

This command allows the user to move one end of a line to a new position.

When the line is selected, the end that is closer to the selection is offered as the end to be moved. The user may request to move the other end instead by entering a backspace character.

The next selection by the user specifies the new location for the end which is to be moved.

Let end-1 be the end specified by the first word of the line buffer, and end-2 be the end specified by the second.

If end-2 is to be moved, the second word of the buffer is replaced by the vector from end-1 to the selected position.

If end-1 is to be moved, then the second word of the buffer is replaced by the vector from the selection to end-2, and the first word is replaced by the vector from the lower left corner of the text of the statement to the selection.

The other vector editing commands are implemented similarly.

B. View Control

1. Jumps and Links

The jump and link machinery is used to select statements to be displayed at the top of the text-viewing area of the screen. Generally speaking, jumps are made within a file and links are used either within or between files. Jumps may be made relative to the structure of the file, to specific statements, or relative to the jump or link ring. Links are to a dynamically determined location in a particular user's file, and can specify that display parameters are to be set when the link is taken.

The jump ring represents the chronological history of the last five jumps made within the current file. Each entry in the ring contains the PSID of the display-start statement and a word representing the display parameters.

The link stack represents the last few links that have been made, and is only updated if the link is to a statement in another file. The entries in this stack contain the user's number, the file name, the PSID of the display-start statement, and a word representing the

display parameters.

Code written in the content-analyzer SPL is used to locate and parse links. The four optional fields of the link are:

user name

file name

location within the file

display parameters.

In parsing a link, those fields which exist are delimited by pointers, which are subsequently used by routines to effect the link.

2. Sequence Generator

The collection of routines known as the sequence generator is used to generate a sequence of statements starting from a given PSID and governed by the current view parameters.

The sequence generator work area is used to maintain information controlling the sequence. This work area is updated by the sequence generator whenever it is called.

The work area includes the following:

- (1) PSID of current statement
- (2) Maximum and minimum level numbers for statements to be included in the sequence
- (3) Current statement's level
- (4) Address of Statement Vector Work Area (SVWA)
- (5) Address of last cell in SVWA
- (6) Address of current last cell used in SVWA.

If statement numbers are being generated, the statement vector is generated for the statement in the SVWA.

The statement vector is a list of words, starting with the level of the statement and followed by entries containing the position of the statement in the

corresponding plexes.

For example, if the statement vector contains (4,1,5,3,2) then the statement is at level four and has statement number 1e3b.

Once the work area has been initialized, the following algorithm is used to determine a candidate for the next statement in the sequence:

If keyword reorganization is being used, then the next PSID can simply be read from a file block.

If a trail is being followed and the current statement contains the appropriate trail marker followed by the name of a statement in the current file, then:

If the statement points to itself then the sequence is terminated by returning a -1;

Otherwise the PSID of the statement pointed to by the trail is returned.

If the current statement has a substatement which is within the current level bounds, then its PSID is returned.

If the current statement has a successor statement which is within the level bounds, then its PSID is returned.

Otherwise, a -1 is returned to indicate the end of the sequence.

After a candidate statement has been selected in the above manner, it must be checked against the current content-analyzer pattern if the content analyzer is in use. If the analyzer is not being used, then the candidate is automatically accepted.

Flags in the ring element for the statement indicate whether the statement has been tested for the current pattern and whether it passed.

If the statement has not been tested, then the sequence generator calls the code compiled for the pattern to make the test. This code is similar to that described for the content-analysis SPL in a previous section. The general flag is set true if the statement passes the pattern, and

false if it does not.

The process of selecting candidate statements is continued until (1) a statement passes the pattern or (2) the sequence is exhausted.

One of the primary uses of the sequence generator is in determining statements to be displayed.

3. Display Parameters

The user has at his disposal two types of display parameters: those which control the selection processes employed by the sequence generator, and those which control the format of the display.

The format parameters control such things as the following:

- (1) The number of lines on the screen
- (2) The position of various viewing areas on the screen
- (3) The size of the characters
- (4) Whether or not the name, number, or signature of a statement is displayed
- (5) The number of lines per statement which are displayed
- (6) Whether or not indenting is used to indicate the structure of the file
- (7) Whether the file is displayed as text or as a tree (schematic).

The selection parameters control the following:

- (1) Whether content analysis is used
- (2) Whether keyword reorganization is used
- (3) Whether a trail is followed
- (4) Whether frozen statements are displayed

(5) Whether the view is limited to only one branch

(6) To what extent the depth into the ring structure is limited.

With the exception of the display parameters which control such things as character size and location of viewing areas on the screen, the display parameters may be modified at any point in the specification of a command.

At certain points in the specification of some commands, the user is given the opportunity of changing the display parameters as part of the command. At other times the user may change them by using Case-3 keyset characters, which are not interpreted as part of a command specification. Furthermore, the availability of a display parameter which causes the display to be regenerated allows the user to treat the changing of display parameters as a pseudo-command. This can be done in the midst of specifying a normal NLS command.

4. The User's Content Analyzer

The user's content analyzer is essentially a subset of the programmer's content-analysis SPL, described elsewhere in this appendix. It is composed of two parts: a compiler and the code which is the product of the compiler.

The compiler is called by a user command to compile content-analysis code from a "pattern" written as text in the user's file (the syntax is that of the content-analysis SPL).

A display parameter then determines whether or not the sequence generator is to execute this code for each of the statements which have passed all other selection criteria.

If executed, the code scans the given statement searching for the specified content. If the search is successful, the statement is displayed; otherwise, it is not.

5. Keyword System

The keyword system provides a rudimentary form of information retrieval in NLS. The result of a keyword search is a list of PSID's. This list is stored in the

keyword file block. The following special terms are used in documenting the keyword system:

hit -- keyword that has been selected and has nonzero weight

result -- one of the PSID's generated by KEYWORD EXECUTE

a. Keyword File-Block Format

The keyword data consists of two tables:

The first contains the PSID's of hits and their weights, with the PSID in the lower 11 bits and the weight in the upper 13.

The second contains the results of the most recent search as an ordered list of PSID's.

The first few words of the block contain information regarding the current status of these tables, such as the following:

- (1) Address of start of second table
- (2) Address of item in second table last returned by the sequence generator to create display
- (3) Address of last entry in second table
- (4) Number of hits.

b. Generation of Results

The following algorithm is used to generate a list of results, given a set of selected keywords.

A table is built with an entry for each result. Each entry takes two words, the first being the hash for the name of the statement, the second the score for the result (i.e., the sum of the weights for all hits referencing that result). The table is generated in the following manner.

For each hit, the statement specified by that PSID is searched for a certain string, which is currently set to be an asterisk followed by two spaces. This search is done by the

content-analyzer POP that does unanchored scans. If the string is not found, then the next hit is considered.

If the string is found, the algorithm then finds the names in the remainder of the statement. Each name is copied out of the text into the statement name register (STN). The algorithm then generates the hash for the name. This is compared to the previous entries to see if it already occurs in the table. If it does, then the score is increased by the weight of the current hit; otherwise, a new entry is created with score equal to the weight of this hit.

After the entries have been accumulated in the above manner, the table is sorted according to score.

The sorted entries are used to produce a list of results. The results are PSID's, so for the hash of each entry, the associated PSID must be found by searching the ring.

Finally, the information at the front of the file block containing the results is updated to show the new number of results.

This list of PSID's is used by the sequence generator when keyword reordering is called for by the user.

6. Text Display

a. General

The collection of routines known as CREATE DISPLAY is used to display in the text area of the user's screen those statements which are selected from the current file by the sequence generator.

The statement selection process and the format of the display are under the user's control by means of VIEWSPECS and the "viewchange" command.

CREATE DISPLAY is called each time the user modifies his file, changes format parameters, selects a new candidate statement for the top of the text area, changes the statement selection parameters, or explicitly requests

that the display be recreated.

A call to CREATE DISPLAY does not imply that the entire display will be recreated. In fact, as little is done as possible in order to minimize file I/O.

The entire display is reconstructed from the display-start PSID only in the following cases:

- (1) A change in the display-start PSID (caused by jumps, "load file" command, etc.)
- (2) Editing involving structural elements larger than statements
- (3) Changes in format parameters
- (4) Explicit user command recreate display.

For statement-editing display changes, the display is updated only for those statements which have changed.

The display recreation is guided by the format parameters, such as truncation, and the output of the sequence generator, which is called to find the first statement in the sequence and for subsequent statements until (1) the last in the sequence has been encountered, or (2) the text area of the screen is full.

b. Implementation Details

The main data areas used by CREATE DISPLAY are the following:

- (1) The display list
- (2) The display list reference table (DLRT)
- (3) The display buffers.

The entries in the display list are used by the display hardware and have the form of a word count followed by a buffer address. The display hardware processes the specified number of words from the buffer pointed to by the entry.

For each line displayed in the text area, there are two entries in the display list.

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. IV: Command Algorithms

The first points to a one-word buffer (that is part of the DLRT entry for that line) that specifies the position of the start of the line on the screen.

The second points to a buffer that contains the actual character string that makes up the line.

For each line there is a four-word entry in the DLRT, containing information such as the following:

- (1) A T-pointer for the first character in the line
- (2) The first and last column numbers containing text in the line (used in bug selection)
- (3) The position on the screen of the left end of the line
- (4) Flags denoting such things as the following:
 - (a) The line is null
 - (b) The line contains special (nonprinting) characters
- (5) A copy of the second display-list entry for the line (used to restore the display list after displaying an error message).

For each PSID which is returned from the sequence generator, a display buffer, DLRT entries, and display-list entries are created.

On the basis of the above description, the actions of CREATE DISPLAY should be clear for cases where the entire text area is being recreated.

The series of statements determined by the sequence generator, starting from the statement specified for the display top, is used to fill the lines of the display, with the appropriate information being stored in the display list, DLRT, and display buffers.

In the case of text-editing changes, the display is only partially recreated; the process is as follows:

The DLRT and display-list entries for the statements that were not edited are copied to auxiliary buffers.

If the content-analyzer flag is off or the edited statement passes the pattern, then a new display buffer, DLRT, and display-list entries are constructed for it.

When this is completed, the DLRT and display list are replaced by the auxiliary buffers and CREATE DISPLAY returns.

c. Bug Selection

It is appropriate to consider the problem of converting selections made by the user to valid character and statement specifications at this point, since bug selection makes use of data areas constructed by CREATE DISPLAY.

Whenever input is read from the user work station, the coordinates of the bug are saved along with it. In the case where the input is meant as a selection by the user, the coordinates must be used to identify a character on the screen. The DLRT contains the information required to do this.

The text area is "homogeneous," in that each line takes a fixed amount of space vertically and each character takes a fixed space horizontally.

Thus the coordinates of the selection can be easily converted to a character and line position in the text area.

This is only part of the problem, however, since the selection may be at a character position that does not contain a character. In other words, there are null areas in the text area and selections in these areas must be "rounded" to another position.

This rounding process is done using the information in the DLRT.

The DLRT has a flag indicating whether a line is null. These flags are checked and the selection moved up the screen until it is on a non-null line.

The DLRT also specifies the first and last columns in the line containing a character. On this basis, the selection is moved to the left or right, if

necessary, to put it on a position containing a character.

It is often the case that bug selections must be converted to T-pointers for operations such as editing.

If the line does not contain any special characters, which take up more than one character position in the SDB, the bug selection can be converted into a T-pointer directly from the information in the DLRT.

There is a flag in the DLRT which indicates whether the line contains any special characters, and a T-pointer for the first character in the line.

If there are no special characters, the character count for column k is simply k greater than the count for the first character and is thus obtainable from the T-pointer in the DLRT entry.

If the line does contain special characters, then the number of special characters in the line to the left of the selected character must be determined. Rather than store this value, it is computed directly from the SDB for the statement. This amounts to reformatting the line up to the selected character.

C. Calculator

The calculator gives the NLS user the ability to perform arithmetic operations using numbers selected from the text or entered from the keyboard.

In addition, arithmetic expressions (functions) with named variables may be evaluated with the aid of a small compiler built into the calculator.

The calculator stores numbers internally in a fixed-length decimal notation (currently using sixteen digits to the left of the decimal and seven to the right).

The arithmetic routines work with numbers that have been "unpacked" into an "accumulator," one digit to a word.

The multiplication algorithm will be briefly outlined as an example.

The multiplicand and the product are in unpacked form.

Digits are read one at a time from the low-order end of the multiplier.

The multiplicand is initially "aligned" with the low-order end of the double-length partial product. During the course of the multiplication, they are realigned by "moving" the multiplicand toward the high-order end of the product.

The first step of the algorithm is to zero the partial product.

Then, until all the digits in the multiplier have been processed, the following algorithm is repeatedly executed:

- (1) Read, and convert to the equivalent binary number, up to four multiplier digits at a time, thus forming a composite multiplier digit.

- (2) For each digit in the multiplicand, multiply it (using the hardware binary multiplication) by the composite multiplier digit, and add the result to the corresponding digit in the partial product.

This takes advantage of the unpacked form to allow "digits" in the partial product to take on very large values. Carries out of the partial-product digits are propagated only once, at the end of the algorithm.

- (3) Realign the multiplicand to the left by the number of digits read from the multiplier.

Now propagate the carries in the partial product to finish the multiplication.

The calculator contains a small operator-precedence compiler for arithmetic expressions.

The compiler produces both code to be interpreted and a symbol table of the variables used in the expression. The symbol table grows toward higher addresses, while the code grows from the other end of the same block of memory.

When the user asks to evaluate the expression, the program asks him to supply values for the variables. The user may fix a variable to a particular value and tell the program not to demand a new value for it. When all variables have been given

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. IV: Command Algorithms

values, the code compiled for the expression is interpreted and the result transferred to the "accumulator" of the calculator.

For each variable in the expression, the symbol table contains the following information:

- (1) The name of the variable (as an A-string, so that it can be displayed in the command feedback line when the user is asked to give it a value)
- (2) The current value of the variable
- (3) Flags indicating whether the user should be asked to supply a value for it when the expression is evaluated, and if so whether it has been given a value during the current evaluation.

The code compiled for the expression is made up of the following instruction types:

- (1) Push values on the stack
 - (a) Push identifier (specified by the address of the value to be pushed)
 - (b) Push constant (the value of the constant follows the instruction in the code)
- (2) Perform arithmetic operations with values on top of stack (unary minus, add, subtract, multiply, and divide)
- (3) Halt

The interpreter for the code simply manipulates the stack and calls the appropriate arithmetic routines.

D. Processors

1. File Cleanup

The file cleanup program serves to verify (and perhaps even restore, with a bit of luck) the internal soundness of an NLS file.

The program goes through the following stages:

- (1) For each structure block:

Set all the name hashes to zero.

Check the free list and mark elements on the free list by setting their hashes to 1.

Verify the used cell count for the block.

(2) For each text block:

Check the free space pointer.

Check each SDB by doing the following:

Compare the length given in the first word of the header to the character count.

Check that the last character is really an end character.

Check that the name character count is reasonable.

Mark SDB's that pass these tests by "OR"ing 36000000B into first word.

If the SDB fails any of the tests, then move the free space pointer up to that point and give up on the rest of that block.

(3) For each graphics block:

The process is similar to the process for text blocks.

At the end of these stages the entire file has been inspected once. During this a special routine has handled the loading of file blocks. If at any time there is a "bad" file block (i.e., one that contains an error), it tries to recover by changing the type of the block if that is in error and recalculating the checksum if that is in error.

File cleanup now continues with a second pass.

(4) Check the actual structure of the ring.

Start from the origin and work through, not trusting the head and tail flags. This requires keeping a stack of father PSID's and comparing each successor to the father.

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. IV: Command Algorithms

Mark ring elements that are used in the structure by setting their hashes to 2 (first making sure that their names are zero, meaning unused, and not one, meaning on the free list).

Mark data blocks (both SDB and VDB) of ring elements in the structure, as used, by changing the top six bits in the first word to 34B instead of 36B.

Correct errors in head and tail flags if any are found.

Errors in structure are handled as follows:

If the bad statement is the head of a plex, then that plex is discarded.

Otherwise the remainder of the plex is discarded.

This discarding is done by linking together good parts of the ring.

Thus in the first case the father of the bad statement simply no longer has any substructure.

In the other case the last good member of the plex becomes the tail of the plex.

If a statement that has valid structure has a bad data block associated with it, then a dummy SDB is created for the statement and file cleanup continues.

(5) Look for "lost" SDB's and ring elements.

Ring elements that still have name hashes of 0 are neither on the free list or in the structure. These are now put on the free list.

SDB's that still have 36000000B in their first word are not pointed to by any statement. These are now marked as garbage.

Marks on SDB's are now erased.

(6) The name hashes for all ring elements in the structure are now recomputed.

This completes the cleanup of the file.

2. File Compaction

The basic objective of the file compactor is to reduce the number of SDB blocks in a file by combining the contents of these blocks and eliminating resultant empty blocks. In addition, empty spaces in the random file are eliminated by packing the file into contiguous blocks. Structure blocks are not compacted.

SDB blocks with fewer than a fixed number of unused cells are not processed -- thus compaction for files which need little or no compacting will be a relatively quick operation.

3. Output Processor

The Output Processor is used to produce hard copy from NLS files. The output of this process includes formatted files for a printer, a Dura typewriter, and a Stromberg-Carlson microfilm machine.

The format of the output is controlled by means of directives.

These are parameters for numerous variables such as page dimensions, page numbering, and "on/off switches" for a large set of format options. The user may control these parameters by means of special strings of text (i.e., output-format commands) embedded in the file text. These command strings, which are also called "directives," are normally suppressed from the hard-copy output.

A full set of directive default values for each type of device has been established; these values may be overridden by directives imbedded in the text of the file.

The Output Processor runs as a subprocess of NLS and has one page -- a buffer -- in common with it. This process, like the compilers, utilizes the statement-selection mechanisms of NLS to obtain its input data. Thus level clipping, content analysis, keyword reordering, trails, and so forth can be used to control what is output via the Output Processor.

4. Compilers

The languages developed by ARC for internal use are

Appendix D: TECHNICAL DESCRIPTION OF NLS
Sec. IV: Command Algorithms

discussed in the main body of this report. Source code for any of these languages may be written in an NLS file and output directly from NLS to the appropriate compiler.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP N/A	
3. REPORT TITLE COMPUTER-AUGMENTED MANAGEMENT-SYSTEM RESEARCH AND DEVELOPMENT OF AUGMENTATION FACILITY			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Final Technical Report 8 February 1968 - 8 February 1970			
5. AUTHOR(S) (First name, middle initial, last name) Dr. D. C. Engelbart and Staff of Augmentation Research Center			
6. REPORT DATE 8 April 1970		7a. TOTAL NO. OF PAGES 284	7b. NO. OF REFS 13
8a. CONTRACT OR GRANT NO. F30602-68-C-0286		9a. ORIGINATOR'S REPORT NUMBER(S) Final Report Project 7101	
b. PROJECT NO. 0967		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) RADC TR-70-82	
c.			
d.			
10. DISTRIBUTION STATEMENT Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.			
11. SUPPLEMENTARY NOTES Monitored by D. Stone AC 315 330-2600 RADC (EMBIH), GAFB, NY 13440		12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency Washington, D.C. 20301	

13. ABSTRACT

This report covers two years of research in a continuing program in the Augmentation Research Center (ARC) of the Information Sciences Laboratory of Stanford Research Institute, supported by ARPA and RADC under Contract F30602-68-C-0286. Some of the work reported was also supported by ARPA and NASA under Contract NAS1-7897.

The research reported is aimed at the development of on-line computer aids for increasing the performance of individuals and teams engaged in intellectual work, and the development of techniques for the use of such aids. The report covers hardware and software development; applications in several areas relating to management of a community of workers who use on-line aids and to information management for such a community, participation in the ARPA computer network, and a summary of plans for the continuation of the research.

KEY WORDS

LINK A

LINK B

LINK C

ROLE

WT

ROLE

WT

ROLE

WT

Computer Augmentation
On-Line Interaction
Management Research

